Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de la Telecomunicación

Servicio web para la recogida de información personal de contexto desde móvil

Autora: Emma Camacho Raya
Tutora: Laura María Roa Romero

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de la Telecomunicación

Servicio web para la recogida de información personal de contexto desde móvil

Autora:
Emma Camacho Raya

Tutora:
Laura María Roa Romero
Catedrática, Universidad de Sevilla

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Trabajo Fin de Grado: Servicio web para la recogida de información personal de contexto desde móvil

Autora: Emma Camacho Raya

Tutora: Laura María Roa Romero

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

A mi familia y amigos

A mis maestros

Agradecimientos

En primer lugar quiero agradecer a mi tutora Laura María Roa Romero, por darme la posibilidad de llevar a cabo este proyecto y a Jorge Calvillo Arbizu por toda la ayuda prestada, guía en el trabajo realizado, y a su disponibilidad y diligencia en todo momento para cualquier duda que apareciera.

En segundo lugar me gustaría agradecer a Javi por todo su ánimo, sus consejos y experiencias, y a mi hermana María por su apoyo en el día a día.

Finalmente gracias a mi familia y amigos que siempre han estado ahí.

Emma Camacho Raya

Sevilla, 2016

Resumen

El ámbito sanitario y el de las tecnologías cada vez están más relacionados. Con el rápido desarrollo de las tecnologías de la información y la comunicación (TIC) surgen nuevas facilidades en el campo de la medicina. Estas ayudan a acortar la distancia entre médicos y pacientes, al permitir, entre otras posibilidades, consultar a los profesionales socio-sanitarios sin necesidad de desplazamientos. Así tenemos la e-Salud (salud electrónica) y la m-Salud (salud móvil) las cuales hacen uso de las TIC para proporcionar nuevos servicios en la sanidad.

Los dispositivos móviles, cada vez más introducidos en nuestra sociedad, ya disponen de ciertos sensores para conocer el contexto del usuario, tanto de su estado fisiológico como de su entorno. El presente trabajo investiga las posibilidades que estos sensores proporcionan para conocer la información de contexto tanto del paciente como del profesional socio-sanitario (por ejemplo, su posición, movimiento, proximidad…), siendo un recurso cada vez más utilizado en el ámbito de la ingeniería biomédica para proporcionar servicios personalizados y adaptados a la situación del usuario en cada instante.

En este trabajo se diseña y desarrolla un servicio web que almacena en una base de datos la información de contexto enviada por smartphones para su posterior uso en la provisión de servicios biomédicos personalizados y conscientes del contexto.

Ante el enorme el abanico de posibilidades que este tipo de servicios de contexto proporcionan, en este trabajo se implementan como ejemplo una aplicación de ayuda a la monitorización de: la localización para pacientes con deterioro cognitivo, la actividad para pacientes con obesidad o problemas cardiovasculares y de condiciones medioambientales del hogar para pacientes de avanzada edad o recién operados. Se hará uso de estos servicios generando alertas personificadas según el contexto del médico en el instante en el que se vaya a mandar la alerta, pudiendo así controlar el método de envío de la alarma según la localización del profesional, modo de audio para llamadas entrantes, estado de la pantalla, etc.

Índice

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En este primer capítulo se expondrán los diferentes objetivos y motivaciones que han servido de precedente a la realización de este proyecto. Además del plan de trabajo seguido para su desarrollo.

1.1 Motivación y objetivos

Desde la invención del telégrafo en 1833 las tecnologías de la información y la comunicación (TIC) han evolucionado de manera rápida y mundial, hasta nuestros días en los que se han introducido completamente en la sociedad añadiendo un amplio conjunto de posibilidades para la comunicación y el intercambio de información. Uno de los campos que se han visto más influenciados por la aplicación de las TIC ha sido el de la asistencia socio-sanitaria. La práctica clínica gira alrededor de datos, información y conocimiento. Internet se ha convertido en la mayor fuente de información sanitaria no solo para los profesionales sino también para los pacientes. Así se tienen ejemplos como la consulta online de tratamientos o historias clínicas.

Dentro del ámbito socio-sanitario, en la última década han surgido diversos paradigmas de aplicación de tecnología como la e-Salud (salud electrónica) y la m-Salud (salud móvil) que prometen revolucionar la práctica socio-sanitaria y satisfacer la creciente demanda de servicios avanzados. Tanto la e-Salud como la m-Salud hacen uso, entre otras, de tecnologías de dispositivos portables y wearables para la provisión de servicios avanzados facilitando la movilidad de los usuarios más allá de los hospitales. Una de las principales aplicaciones de este tipo de tecnologías es la monitorización de usuarios, tanto de su estado fisiológico como de su entorno.

Algunas posibilidades de monitorización incluyen la obtención de información de datos personales como: la tensión arterial, el índice de masa corporal, el peso corporal, las horas de sueño y las kilocalorías consumidas a lo largo del día, o la monitorización de la posición de individuos con deterioro cognitivo, monitorización de condiciones ambientales y monitorización de actividad. La monitorización remota de pacientes permite también disponer de datos continuos de sus variables fisiológicas frente al seguimiento sanitario tradicional donde la recogida de información del paciente se realiza exclusivamente cuando este se encuentra en presencia física del médico.

Además de la sensorización de condiciones ambientales y fisiológicas para prestar servicios socio-sanitarios, en escenarios móviles, el acceso a los propios servicios debe tener en cuenta la situación del usuario y las características del dispositivo de acceso, es decir, el contexto de uso. Adaptarse al contexto del usuario en cada instante garantiza una mayor experiencia de usuario y aumenta la usabilidad de los servicios. Así es necesario la personalización y adaptación al contexto del paciente y del profesional socio-sanitario, conocer si el dispositivo se encuentra apagado, la localización del profesional (por si se encuentra en su lugar de trabajo), obtener información de la proximidad, luminosidad, red y todo lo necesario o posible desde el dispositivo portable. A partir de este contexto se podrá personalizar el tipo de alarma que recibirá, ya que por ejemplo si se recibe una alarma al móvil y éste está apagado, la alarma se pierde y habría que buscar una alternativa.

El contexto del usuario (tanto personal como ambiental) puede ser monitorizado a través de sensores portables o incluidos en el escenario, y los avances en este tipo de tecnologías y la disminución de costes de producción están permitiendo que se incluyan en todo tipo de dispositivos (por ejemplo, sensores para domótica, wearables, relojes y teléfonos inteligentes, etc.). Sin duda, los dispositivos más versátiles a la hora de incorporar sensores son los smartphones, cuya alta penetración los está posicionando como un dispositivo de propósito general para el acceso a un amplio conjunto de servicios basados en movilidad. Hoy en día millones de personas portan de forma continua un dispositivo móvil, los cuales cada vez ofrecen más funcionalidades debido al rápido aumento de potencia de sus procesadores. Por ejemplo, España es el cuarto de los países desarrollados en penetración de smartphones, con un 85% [1] (Figura 1). Estos smartphones cada vez más inteligentes ya disponen de sensores con los que podemos obtener medidas, movimientos e información sobre el contexto de los usuarios.

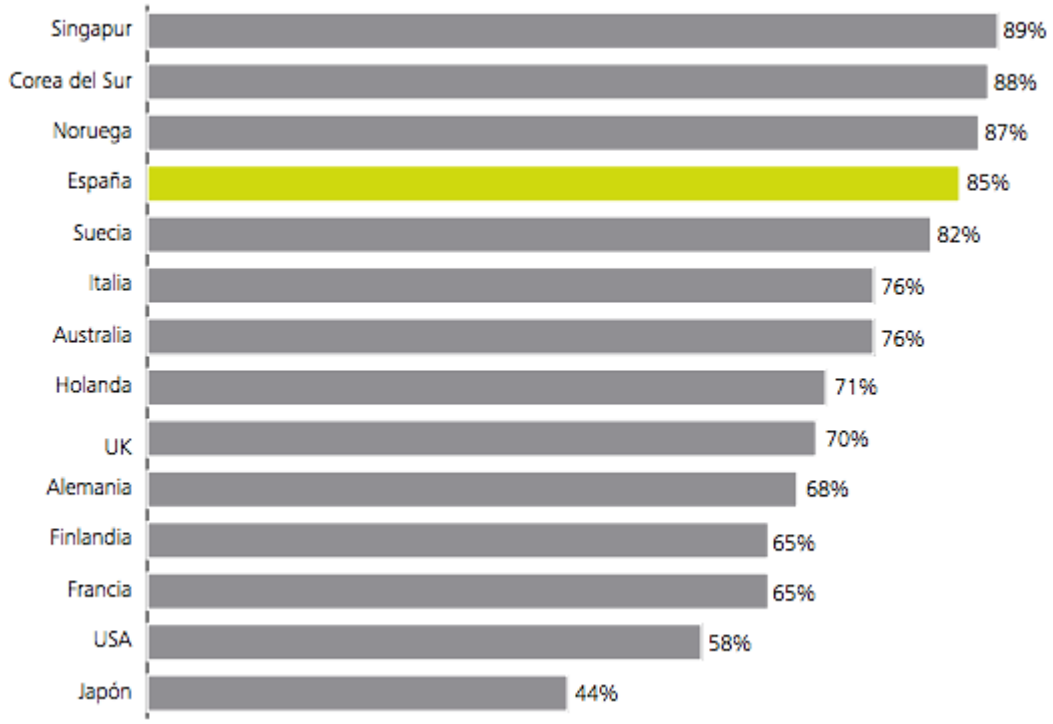


Figura 1. Penetración de Smartphone en países desarrollados

Los smartphones, ya sea a través de los sensores que integran o como intermediarios de otros dispositivos de monitorización, poseen un enorme potencial para facilitar un seguimiento del paciente a distancia y en tiempo real. En este sentido, las TIC podrían constituir un método prometedor y plantearse como una alternativa a los tratamientos tradicionales para reducir costes económicos, disminuir el tiempo entre visitas y tener un mayor control del paciente.

Por todo ello, la hipótesis de este trabajo es la posibilidad de monitorizar las condiciones ambientales y fisiológicas de una persona a través de un smartphone, almacenando dicha información de contexto para utilizarla en la provisión de servicios personalizados.

Para verificar dicha hipótesis, este trabajo tiene como objetivo principal investigar la recogida, almacenamiento y distribución de información de contexto de usuarios (ciudadanos, profesionales socio-sanitarios, pacientes, etc.) a través de un servicio web para proporcionar servicios personalizados y adaptados a la situación del usuario en cada instante. Como elementos de sensorización de la información de contexto se partirá de los sensores comúnmente incluidos en cualquier smartphone.

Este objetivo principal se puede descomponer en los siguientes subobjetivos:

1. Estudio de los sensores incluidos en los smartphones
2. Diseño y desarrollo de una base de datos para almacenar la información de contexto
3. Diseño y desarrollo de un método para el almacenamiento y recuperación de dicha información
4. Diseño y desarrollo de servicios de ejemplo que hagan uso de dicho método y la información de contexto de usuarios para la provisión de servicios personalizados.

Como servicios de ejemplo se desarrollarán los siguientes:

- un servicio de monitorización de localización para pacientes con deterioro cognitivo. Dependiendo del grado de avance de la enfermedad, una persona puede ser consciente de sus actos y recordar hechos pasados sin ningún problema y a veces sufrir desorientación. En casos más extremos, el enfermo pierde la memoria en intervalos de tiempo sin ser consciente de sus hechos anteriores [2]. Así se propone un servicio que usando una localización y un radio se cree un perímetro prefijado en el que se controle si el usuario entra o sale de este, generando una alerta.
- un servicio de monitorización de la actividad para pacientes con obesidad o problemas cardiovasculares.

Este servicio controlará el número de pasos del paciente existiendo la posibilidad de generar una alarma al alcanzar un determinado umbral de pasos.

- un servicio de monitorización de condiciones medioambientales del hogar para pacientes de avanzada edad o recién operados. Se controlará que tanto la temperatura como la humedad del hogar se encuentre dentro de unos límites configurados por el profesional. Hay intervalos en los que la temperatura puede ser perjudicial para personas mayores, siendo recomendado que se encuentre alrededor de 21ºC. El frío intenso provoca una vasoconstricción periférica intensa con aumento del metabolismo basal y producción de calor. El calor produce vasodilatación periférica, sudoración abundante, pérdida de agua y electrólitos a través de la piel. En situaciones muy extremas se puede dar el llamado golpe de calor, que causa hipertermia, deshidratación, dolor de cabeza y afectación del sistema nervioso central. Con respecto a la humedad relativa es recomendada que se encuentre entre 40% y 50%. Una humedad relativa (HR) demasiada alta en una vivienda u oficina ocasiona: respiración con dificultad, fatiga, contribuye a la acumulación de calor en el organismo, baja liberación de toxinas por la respiración, los malos olores molestan más. Una HR demasiado baja ocasiona: el polvo levita en el aire con más facilidad lo que acentúa las alergias, proliferan los resfriados, inflamación ocular, asma y dolor de cabeza entre otras dolencias. Esto puede provocar respiración con dificultad. Dependerá principalmente de la ciudad donde se encuentre [3].

Estos servicios son sólo una muestra del potencial que la información de contexto de usuario puede suponer para la personalización y adaptación de los servicios a distintos escenarios y usuarios o la provisión de servicios de m-Salud flexibles en cuanto al contexto.

En este trabajo se diseñará y desarrollará un servicio web que almacene en una base de datos la información de los usuarios. A través de una aplicación se obtendrán los datos del dispositivo y se hará uso de estos servicios generando alertas personificadas según el contexto del usuario (médico, familiar, cuidador, etc.) en el instante en el que se vaya a mandar la alerta.

Este tipo de servicios de provisión de información de contexto son especialmente interesantes en la actualidad por la falta de servicios socio-sanitarios que se adaptan al contexto del usuario. Además, el contexto también es importante a la hora de enviar el dato y fusionarlo con la información almacenada (el contexto permite saber hora de medida, dispositivo de medida, etc.)

1.2 Plan de trabajo

El plan de trabajo de este proyecto sigue un desarrollo en cascada con las siguientes etapas:

- Documentación: En esta etapa incluimos la investigación de los servicios web, de trabajos y aplicaciones anteriormente realizados, los distintos sensores de los que puede disponer un dispositivo móvil y la documentación de Android para la realización de la aplicación de ejemplo.
- Estudio del alcance: Como cada proyecto es necesario tener una visualización del alcance y los pasos necesarios para llegar hasta nuestro objetivo. En esta etapa se obtendrá una idea del resultado del proyecto así como los pasos hasta conseguirlo.
- Diseño: Aquí se realizarán los diseños de las distintas partes del servicio web:
    1) Primero la base de datos con todas las tablas y sus relaciones.
    2) Las distintas secuencias SQL necesarias para interactuar con la base de datos.
    3) Conocer el método con el que conectaremos con el servidor y la base de datos, y el lenguaje en el que recibiremos los datos.
    4) Diseño de la aplicación Android de ejemplo, incluyendo la interfaz de usuario, y los distintos servicios y clases que utilizará.
- Implementación: En esta etapa es donde se desarrolla toda la arquitectura del servicio web y la aplicación:
    1) Implementación de la base de datos.

2) Codificación de los distintos scripts que usará el servicio para la comunicación entre el servidor y la aplicación
3) Desarrollo de las distintas partes de la aplicación: la interfaz de usuario, las clases y los servicios necesarios.

- Validación: Como etapa final se realizarán pruebas para el correcto funcionamiento, una vez que se tiene un primer modelo del servicio, siguiéndole una corrección de errores y realización de mejoras en las que se cumplan la mayoría de requisitos posibles programados al comienzo del desarrollo del proyecto.

2 FUNDAMENTOS TÉCNICOS

En este capítulo se dará paso a la descripción de la importancia del contexto en la asistencia socio-sanitaria haciendo hincapié en la problemática de la introducción. Además, se detallarán los distintos sensores dentro de un smartphone así como algunas aplicaciones del ámbito biomédico existentes en el mercado. Se realizará una introducción a los servicios web y se presentarán los entornos de desarrollo utilizados para el proyecto, explicando cada uno de los elementos que intervienen. El capítulo concluye con la metodología usada para la realización del trabajo.

2.1 La importancia del contexto en la asistencia socio-sanitaria

La e-Salud y las nuevas tecnologías han comenzado a utilizarse con éxito en el mundo sanitario. Sus ventajas: permiten mejorar la práctica clínica diaria gracias a la mejora en la búsqueda de información, seguridad en la prescripción, archivo de datos, comunicación entre profesionales, etc. Además, permiten optimizar los procesos mejorando la eficiencia y ahorrando costes.

Dentro de la e-Salud se incluye la administración digital de historias clínicas que facilita el archivo, consulta, edición e intercambio de datos de los pacientes entre diversos profesionales sanitarios -centros de salud, hospitales, especialistas, farmacias-, la prestación de servicios de salud utilizando las tecnologías de la información y la comunicación, especialmente donde la distancia es una barrera para recibir atención de salud, la aplicación de las tecnologías de la información y la comunicación al aprendizaje, la m-Salud, término empleado para designar el ejercicio de la medicina y la salud pública con apoyo de los dispositivos móviles, como teléfonos móviles, dispositivos de monitoreo de pacientes y otros dispositivos inalámbricos. La monitorización de usuarios ofrece múltiples facilidades, principalmente conocer variables fisiológicas en el día a día de un paciente y así disminuir el tiempo entre visitas y los desplazamientos, también mejorar la investigación de la evolución y los factores que afectan a una enfermedad para poder prevenirla y mejorar su cuidado.

La atención socio-sanitaria abarca un amplio conjunto de escenarios de asistencia como el tratamiento del cáncer, enfermedades crónicas, embarazo, obesidad, enfermedades de la piel, personas mayores viviendo solas y enfermedades mentales. Investigaciones previas plantean la necesidad de prevenir y diagnosticar estas situaciones clínicas lo más pronto posible a través del conocimiento del día a día de los pacientes, como sus estados de ánimo, lo que comen, sus horas de sueño, etc. Un ejemplo de ello lo encontramos en [4] que propone un servicio de atención sanitaria para la monitorización de la salud en personas mayores considerando 6 parámetros diferentes: 1) actividad física, 2) presión arterial, 3) glucosa, 4) medicación, 5) frecuencia cardiaca y 6) peso. La monitorización puede aumentar su relevancia incluyendo además condiciones ambientales que pueden influir en el usuario a nivel de salud como la temperatura o la humedad del hogar. En [5] se diseña un servicio web considerando tanto la información de contexto del usuario como la información meteorológica relacionada con enfermedades, como el asma o las alergias. En general, en el caso de personas mayores viviendo solas o pacientes recién operados es interesante monitorizar el hogar para asistirle con tareas diarias o enviar alarma si sucede algo, tanto a nivel personal como ambiental.

Con los rápidos avances de las TIC los nuevos móviles permiten el acceso a servicios de cualquier tipo, siendo posible tener una visión del contexto en el que se encuentre un usuario y proporcionar servicios adaptados. Los teléfonos móviles suponen una excelente plataforma para el despliegue de servicios ubicuos sensibles al contexto, ya que estos han pasado a ser un objeto cotidiano, un elemento prácticamente imprescindible en la vida diaria y que nos acompaña en todo momento. Por otra parte, este aspecto se ha acentuado en los últimos

años, pues los notables avances tecnológicos han convertido los teléfonos móviles en potentes computadores de bolsillo, capaces de realizar multitud de tareas y de estar comunicados en todo momento. Este uso multidisciplinar y permanente de los dispositivos móviles, unido a los múltiples sensores que actualmente incorporan, hacen que la cantidad de información de contexto que pueden proporcionar sea notable.

Esta información de contexto puede proporcionar tanto el estado del dispositivo (pantalla encendida o apagada, modo de audio para una llamada, localización, proximidad, luminosidad, etc.) como en el instante de tiempo del contexto. Pudiéndose adaptar en cada instante a esta información para la personificación del servicio.

2.2 Sensores y smartphones

Las condiciones ambientales y fisiológicas de los usuarios pueden ser monitorizadas a través de sensores dedicados, aunque la sofisticación y miniaturización de las tecnologías móviles está posibilitando que los smartphones incluyan este tipo de sensores. Los sensores de los dispositivos móviles se dividen en tres grupos: de movimiento, de medioambiente y de posición. La variedad y poder de monitorización de estos sensores incluidos en los teléfonos móviles los hace interesantes para la provisión de servicios conscientes del contexto del usuario.

Para este trabajo se ha partido de los sensores comunes incluidos en cualquier smartphone [6] [7]. Estos son:

- Acelerómetro: sensor de movimiento con el que se detecta la orientación del dispositivo a través de tres pequeños tubos en cuyo interior, colocado en la parte superior, existe un muelle en cuya punta hay una bola haciendo de masa. Esto quiere decir que si tenemos tres tubos simulando los tres ejes de coordenadas tridimensionales, si movemos este conjunto, la bola se desplazará dentro de los tubos. Así es como sabemos la posición de nuestro dispositivo.

- Giroscopio: sensor de movimiento que nos provee de más y mejor información en cuanto a la posición del dispositivo con respecto al acelerómetro. Es un dispositivo mecánico que nos ayudará a medir, mantener y cambiar la orientación de algún dispositivo basándose en el momento angular.

- Magnetómetro: sensor de posición que permite detectar el polo norte magnético para la brújula o la magnitud de fuerza magnética de un objeto.

- Sensores de proximidad: sensor de posición que se basan en un LED infrarrojo y en un receptor IR que emite una luz infrarroja y si algo se interpone en esta luz la devuelve al receptor detectando así que hay algo muy cerca del dispositivo. En el caso de ser una llamada, será la oreja del usuario.

- Sensores de luz: sensor de medioambiente que convierte la cantidad de luz detectada en una señal eléctrica con la que pueda trabajar nuestro terminal. Así se ajusta automáticamente el brillo de nuestra pantalla.

- Barómetro: sensor de medioambiente que algunos smartphone disponen, mide la presión ambiental.

- Sensor de ritmo cardiaco: este sensor se encarga de recopilar las pulsaciones por minuto que se obtiene con tan sólo colocar el dedo sobre él. Se basa en los cambios de color cuando el dedo está iluminado por el flash LED.

- Termómetro y sensor de humedad: sensores de medioambiente del que dispositivos más modernos disponen para controlar su temperatura interior por si se sobrecalientan y conocer la humedad del ambiente.

- Podómetro: sensor de movimiento que cuenta el número de pasos realizados, algo que con menos precisión puede hacer el acelerómetro.

- Lector de huellas: se basa en sensores capacitivos, que reconocen los dactilogramas y los recopila en una imagen digital que almacenarán para compararla con cada dedo que intente desbloquear nuestro dispositivo. Empezó con el iPhone 5S y hoy en día lo tienen más dispositivos.

- También sería posible que el móvil sirviera como intermediario de la información recogida por otros sensores que el paciente lleva implantados y que no tienen conexión a Internet para enviar la información por sí mismos. El móvil en este caso hace de proxy.

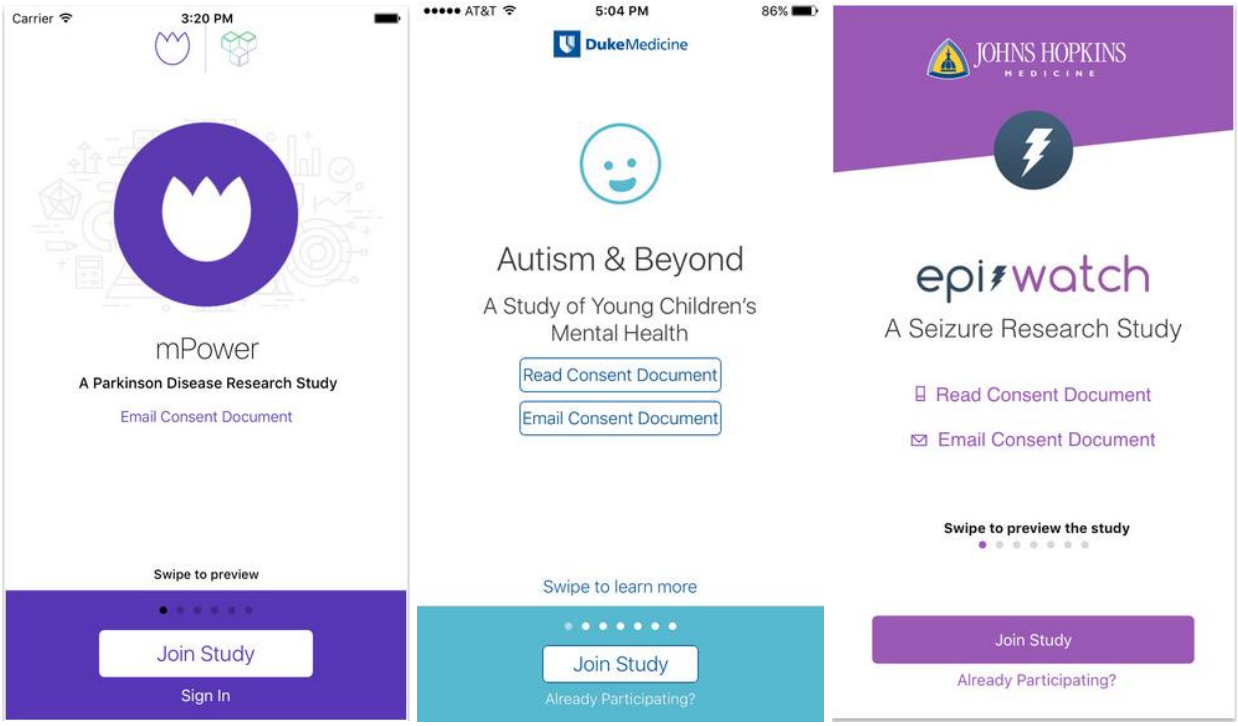
La información de estos sensores puede permitir personalizar o adaptar diferentes tipos de servicios al contexto del usuario. Existen algunas aplicaciones móviles que utilizan estos sensores con diferentes finalidades, destacamos aquí algunas.

2.2.1 ResearchKit

Es un conjunto de aplicaciones de Apple que, aunque no estén en lenguaje Android sino en IOS, son bastante interesantes.

ResearchKit [8] nació con el objetivo de mejorar la investigación médica, recopilando datos de cada uno de los usuarios. Estas son algunas de las aplicaciones (Figura 2):

- mPower: Desde su lanzamiento en 2015, más de 10.000 usuarios se han inscrito en la app mPower para participar en el estudio del Parkinson. Esta app usa el giroscopio y otras prestaciones del iPhone para cuantificar la destreza, el equilibrio, la manera de caminar e incluso la memoria. Los datos recogidos han servido a los investigadores para comprender mejor el Parkinson y para profundizar en factores que influyen en su evolución, como el sueño, el ejercicio físico o el estado de ánimo.
- Autism & Beyond: La app Autism & Beyond utiliza la cámara HD delantera del iPhone para grabar a niños desde los 18 meses viendo una serie de vídeos. Luego analiza sus reacciones emocionales mediante unos innovadores algoritmos de reconocimiento facial. No es necesario que los niños vayan a la consulta para hacer esta prueba, lo que hace mucho más fácil un diagnóstico y un tratamiento más tempranos.
- EpiWatch: Los investigadores esperan que el Apple Watch les ayude a prever las crisis epilépticas antes de que sucedan. La app EpiWatch permite a los usuarios llevar un registro preciso del inicio y la duración de las crisis en tiempo real, lo que crea una correlación entre el historial de crisis y la medicación. En cuanto los participantes notan la convulsión, abren la app tocando una complicación creada a medida para el Apple Watch. En ese momento se activan el acelerómetro y el sensor de frecuencia cardiaca, y se envía un aviso al familiar o médico designado.



a b c

Figura 2. Aplicaciónes de ResearchKit. (a) mPower; (b) Autism & Beyond; (c) EpiWatch.

2.2.2 iCare Monitor de la salud

Aplicación ofrecida por iCare Fit Studio [9] que mide la presión sanguínea, el ritmo cardíaco, la visión, la audición, SpO2 y frecuencia respiratoria solamente por teléfono.

Figura 3. Aplicación iCare Monitor de la salud

2.3 Servicio Web

En esta sección haremos una introducción a los servicios web con su posterior clasificación y elección de la solución.

2.3.1 Historia

La preocupación por los sistemas distribuidos y de cómo diferentes máquinas podían comunicarse entre sí surgió en la década de los 90. Hasta ese momento, era suficiente con que las aplicaciones de un mismo ordenador pudieran establecer una comunicación. Los servicios web surgieron entonces, ante una necesidad de estandarizar la comunicación entre distintas plataformas (PC, Mainframe, Mac, etc.) y lenguajes de programación (PHP, C#, Java, etc.).

En 1990, surgieron los modelos COM y CORBA (Common Object Request Broker Architecture). El primero, fue creado por Microsoft, y el segundo, por el Object Management Group. No obstante, estos dos modelos presentaban un hándicap muy importante: no eran fácilmente interoperables.

Más adelante, Microsoft creó DCOM (Distributed Component Object Model) y Sun, RMI (Remote Method Invocation). Aunque estos métodos permitían establecer una conexión entre ordenadores a través de la red, tampoco eran interoperables ya que RMI está disponible únicamente para Java, y por tanto, es dependiente del lenguaje de programación. Por todo ello, Microsoft empezó a interesarse por la computación distribuida basada en XML en el año 1997. Su objetivo era terminar con los problemas de interoperabilidad de las soluciones anteriores y permitir que las aplicaciones se conectaran mediante RPCs (Remote Procedure Call), utilizando los estándares de comunicación XML y HTTP.

Precisamente, será la búsqueda de esta interoperabilidad un factor clave para el devenir de los servicios web. En una sociedad visionaria que empezaba a cambiar, una sociedad de la información, que demandaba de esta flexibilidad para construir aplicaciones cada vez mayores; eso sí, a partir de componentes distribuidos más pequeños. Tendencia que sigue vigente tras el transcurso de los años.

2.3.2 Definición

Un servicio web es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C (World Wide Web Consortium) son los comités responsables de la arquitectura y reglamentación de los servicios Web.

Los servicios web principalmente pueden estar basados en SOAP o en REST:

- SOAP (siglas de Simple Object Access Protocol):

  Es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambios de datos XML, el punto identificativo de SOAP es que las operaciones son definidas como puertos WSDL (Web Services Description Language). Es por esto que será aconsejable utilizar este protocolo en entornos donde se establecerá un contrato formal y donde se describirán todas las funciones de la interfaz así como el tipo de datos utilizados tanto de entrada como de salida. El lenguaje WSDL nos permitirá definir claramente cualquier detalle de las funciones de nuestro WS.

- REST (Representational State Transfer)

  Es un estilo de arquitectura de software para sistemas distribuidos tales como la web, a diferencia de SOAP, se centra en el uso de los estándares HTTP y XML para la transmisión de datos sin la necesidad de contar con una capa adicional. Las operaciones(o funciones) se solicitarán mediante GET, POST, PUT y DELETE, por lo que no requiere de implementaciones especiales para consumir estos servicios. Además se podrá utilizar JSON en vez de XML como contenedor de la información, por lo que será aconsejable utilizar este protocolo cuando busquemos mejorar el rendimiento, o cuando disponemos de escasos recursos, como sería el caso de los dispositivos móviles.

En este proyecto se usará REST debido a que nuestro objetivo está en desarrollar algo sencillo, practico y que presente buena escalabilidad y rendimiento.

2.3.3 REST

Empezamos aclarando que REST no es un ningún tipo de estándar, aunque está basando en algunos de ellos como pueden ser:

• HTTP

• URL

• Representación de los recursos: XML/HTML/GIF/JSON/JPEG/…

• Tipos MIME: text/html, application/json…

La escalabilidad alcanzada en estos sistemas se debe principalmente a las siguientes pautas que se ha de seguir en su diseño. Éstas son:

• Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión.

• Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Ellos suelen ser comparados con las operaciones asociadas a la tecnología de base de datos, operaciones CRUD: CREATE, READ, UPDATE, DELETE.

• Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso se puede direccionar únicamente a través de su URI.

• Hipermedia como un mecanismo del estado de la aplicación. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

2.4 Materiales

Los elementos más importantes usados para la realización del trabajo son los siguientes.

2.4.1 Android

Android es un sistema operativo inicialmente pensado para teléfonos móviles, al igual que iOS, Symbian y Blackberry OS [10]. Lo que lo hace diferente es que está basado en Linux, un núcleo de sistema operativo libre, gratuito y multiplataforma.

El sistema permite programar aplicaciones en una variación de Java llamada Dalvik. El sistema operativo proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda, etc.) de una forma sencilla en un lenguaje de programación conocido como es Java.

Esta sencillez, junto a la existencia de herramientas de programación gratuitas, hace que una de las cosas más importantes de este sistema operativo sea la cantidad de aplicaciones disponibles, que extienden casi sin límites la experiencia del usuario.

Una de las varias características de este sistema operativo es que es completamente libre. Es decir, ni para programar en este sistema ni para incluirlo en un teléfono hay que pagar nada. Y esto lo hace popular entre fabricantes y desarrolladores, ya que los costes para lanzar un teléfono o una aplicación son muy bajos.

Por todas estas razones y por ser el más usado en España, presente en el 93,9% de los terminales como podemos comprobar en la Figura 4 [11], es en el que se ha decidido centrar el proyecto.

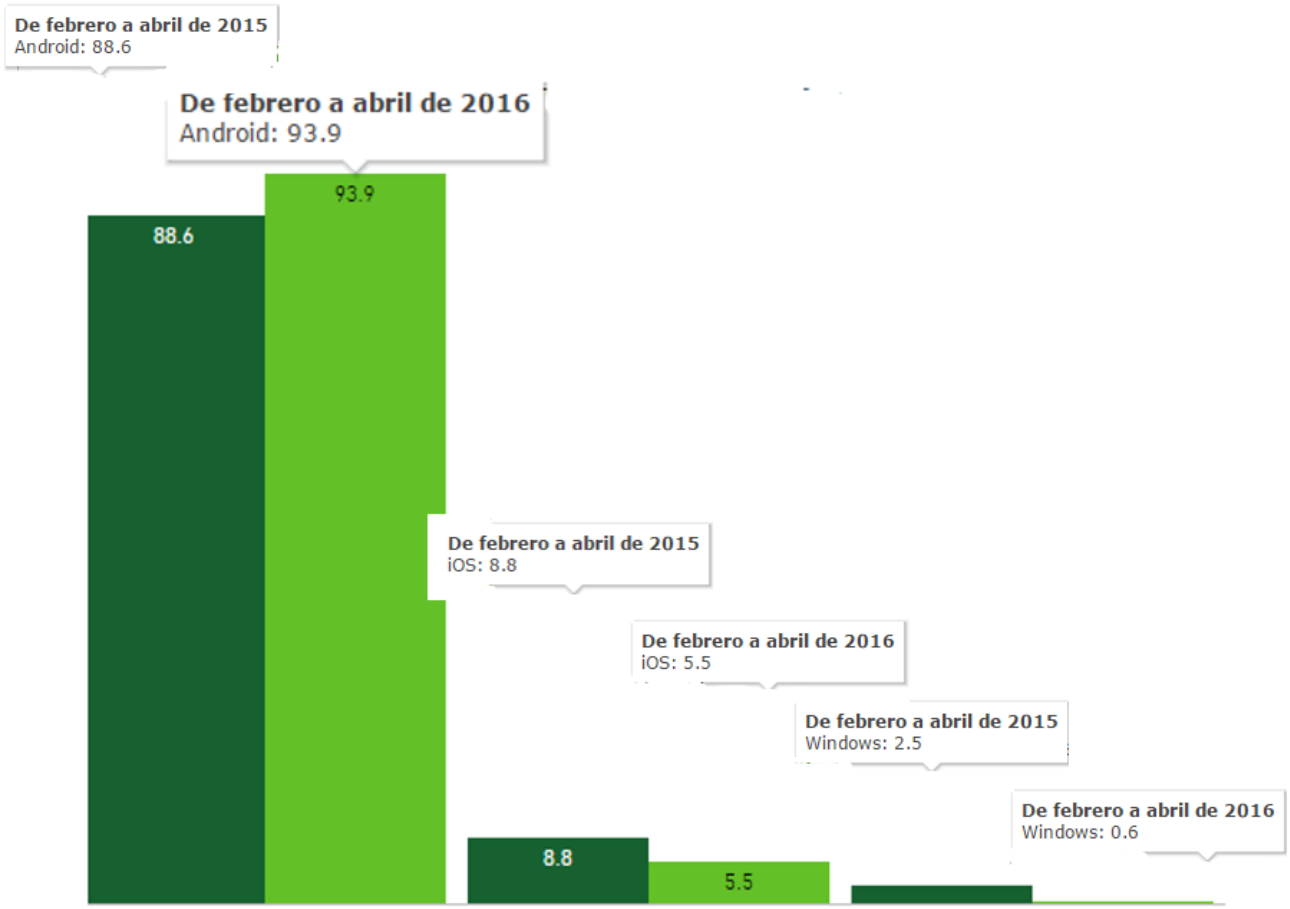


Figura 4. Uso de los distintos sistemas operativos en España

Android ha ido evolucionando a lo largo de los años con nuevas versiones hasta la actual 6.0, llamada Marshmallow. Siendo Lollipop la más distribuida actualmente, como puede comprobarse en la Figura 5 [12], ha sido la elegida para la realización del proyecto, en concreto la versión 5.0, la API (Application Programming Interface) 21.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.2%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.0%
4.1.x	Jelly Bean	16	7.2%
4.2.x		17	10.0%
4.3		18	2.9%
4.4	KitKat	19	32.5%
5.0	Lollipop	21	16.2%
5.1		22	19.4%
6.0	Marshmallow	23	7.5%

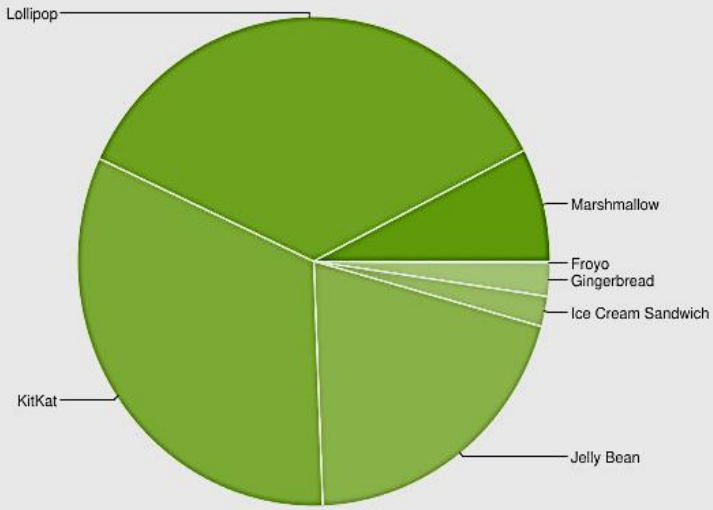


Figura 5. Versiones de Android más usadas actualmente

2.4.2 XAMPP

XAMPP [13] es un servidor independiente de plataforma, software libre, que consiste principalmente en la base de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script: PHP y Perl. El nombre proviene del acrónimo de X (para cualquiera de los diferentes sistemas operativos), Apache, MySQL, PHP, Perl.

El uso que se le da en el proyecto a XAMPP es para emular los servicios necesarios para el correcto funcionamiento de la aplicación de manera local, tales como el servidor Http (Servidor Apache), así como la gestión y administración de la base de datos mediante MySQL. Es importante destacar que XAMPP debe ejecutarse bajo los derechos de administrador para poder funcionar de manera correcta.

Figura 6. Logo de XAMPP

2.4.3 Apache

El servidor HTTP Apache [14] es un servidor web HTTP de código abierto. Apache presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido, soporte para SSL y TLS y soporte para lenguaje PHP. En nuestro proyecto será el encargado de recibir las peticiones POST y GET de la aplicación, manejarlas con los script PHP, y realizar las consultas a la base de datos de acuerdo con unos permisos establecidos previamente en sus ficheros de configuración (httpd.conf) y devolver una respuesta a la aplicación.

Figura 7. Logo de Apache

2.4.4 MySQL

MySQL [15] es un servidor de bases de datos relacionales usado en el proyecto para almacenar los datos de la aplicación de manera remota, donde se introducirán los datos del usuario y en el que se podrán realizar consultas de los datos disponibles. Para poder gestionar esos datos, tan solo se debe poner a funcionar XAMPP, y gracias al botón "Admin", aparecerá directamente PhpMyAdmin, que es un entorno de trabajo para poder configurar la base de datos que se emplea en el proyecto. PhpMyAdmin es una herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas web. En este entorno se podrán introducir las tablas, especificar las relaciones entre ellas, introducir datos, eliminarlos, es decir, realizar todas las operaciones relacionadas con la estructura de la base de datos.

Figura 8. Logo de MySQL

2.4.5 Android studio

Android Studio [16] es el entorno de desarrollo integrado (IDE) oficial de Android. Utilizado para el desarrollo y testeo de la aplicación. Existen otras alternativas como Eclipse y NetBean, pero Android Studio ofrece otras facilidades al ser exclusivo de Android.

Fue presentado por Google el 16 de mayo del 2013 y la última versión es la 2.1 pero se usará la versión 1.4.

Figura 9. Logo de Android Studio

2.4.6 Advanced REST Client

Advanced REST Client [17] es una herramienta que puede encontrarse como extensión para el navegador Google Chrome y, que como su propio nombre indica hace las veces de un cliente REST. Además, ayuda a cualquier desarrollador web a crear y testar peticiones HTTP de prueba.

La iniciativa de esta herramienta nace con la idea de poder ir comprobando gran parte de la implementación del servicio eb, sin necesidad de tener programada la parte de la aplicación Android que se encargue del consumo del servicio web. Presenta un interfaz muy simple, incluso para usuarios que no hayan usado ningún cliente REST anteriormente.

Para testear el funcionamiento del servicio simplemente se introduce la URI correspondiente junto con el método HTTP a utilizar: GET, POST, PUT, DELETE u otros, y los parámetros que lleve la URI.

Figura 10. Logo de Advanced REST client

2.4.7 PHP

PHP [18] es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. En el proyecto se usará para escribir los métodos del servicio web que contendrán las peticiones a la base de datos.

2.4.8 JSON

JSON, acrónimo de JavaScript Object Notation [19], es un formato de texto ligero para el intercambio de datos. JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos. Nació como una alternativa a XML, el fácil uso en javascript ha generado un gran número de seguidores de esta alternativa. Una de las mayores ventajas que tiene el uso de JSON es que puede ser leído por cualquier lenguaje de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías.

Un array de objeto JSON tiene el aspecto que se presenta en la Figura 11.


```
-0:  {
     "Fecha": "2016-05-19 22:18:14"
     "Light": "7"
     "usuario_dni": "31008811e"
 }
-1:  {
     "Fecha": "2016-05-27 12:48:23"
     "Light": "94"
     "usuario_dni": "31008811e"
 }
```


Figura 11. Array de objetos JSON

2.4.9 DBDesigner

DBDesigner [20] es una aplicación web que permite diseñar bases de datos como un diagrama UML, permitiéndote convertirlo posteriormente a un script SQL.

2.4.10 Visual Paradigm

Visual Paradigm [21] es un software de modelado UML que nos permite analizar, diseñar, codificar, probar y desplegar. Dibuja todo tipo de diagramas UML, genera código fuente a partir de dichos diagramas y también posibilita la elaboración de documentos. En el trabajo será utilizado para los diagramas UML.

2.4.11 Recursos hardware

- Samsung GALAXY A5, dispositivo móvil principal con el que se realizó el desarrollo de la aplicación.
- Samsung GALAXY S6, dispositivo móvil en el cual se realizaron pruebas para los distintos sensores.
- Samsung GALAXY TAB A 9.7, dispositivo móvil tipo tableta en el cual se realizaron pruebas.
- Portátil ACER Aspire V15 Nitro, empleado para el desarrollo de todo el proyecto.

2.5 Metodología y arquitectura

Para el desarrollo de este proyecto se optará por una metodología en forma de cascada la cual ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. Al final de cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase.

Así se irá avanzando en las distintas etapas explicadas en el siguiente apartado. Existiendo la excepción de las etapas de implementación y validación, las cuales seguirán una metodología iterativa, ya que se irán desarrollando distintas versiones para ir llevando a cabo cada uno de los requisitos, estos es así porque cada nuevo fragmento de código ha de comprobarse que funciona correctamente y como era esperado

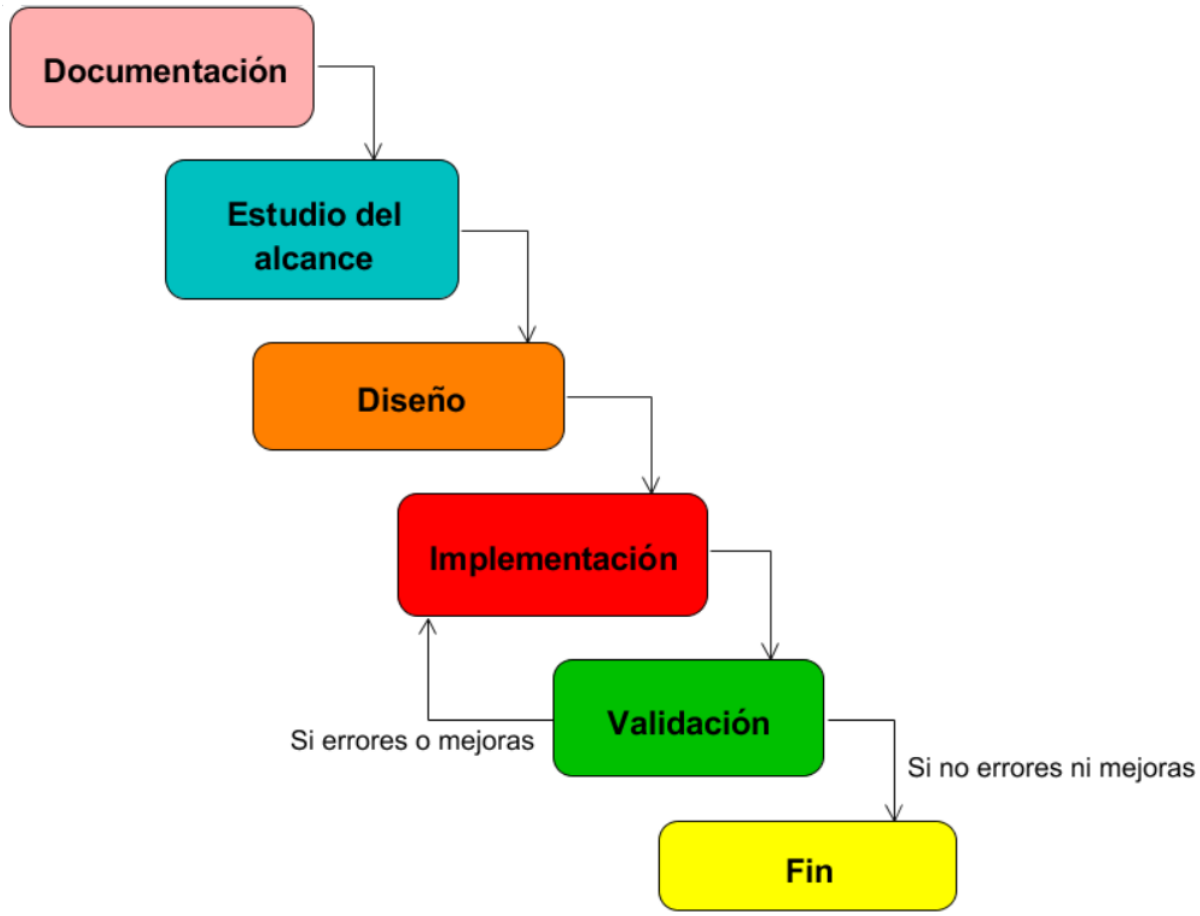


Figura 12. Metodología del trabajo

Se potenciará el uso de tecnologías y herramientas abiertas, que cuenten con una alta penetración en el mercado y madurez, comentadas en el apartado anterior. Así en la etapa del diseño se hará uso de la aplicación web dbDesigner para el diseño de la base de datos en MySQL y Visual Paradigm para el diseño de la aplicación Android de ejemplo. En la etapa de implementación se usará XAMPP con Apache y MySQL para tener el servidor y la base de datos, PHP para el desarrollo de los métodos que usara el servicio web para comunicarse con la base de datos y Android Studio para el desarrollo de la aplicación Android usando JSON como lenguaje en el que se recibirán los datos del servicio web en la aplicación. Para la etapa de validación se probará el correcto funcionamiento del servicio web REST con Advanced REST Client y el de la aplicación Android con los dispositivos móviles de los recursos hardware.

Como arquitectura de la solución se usará la arquitectura cliente-servidor definida como un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, equipos generalmente muy potentes en materia de capacidad de entrada/salida, y los demandantes, llamados clientes. Un cliente realiza peticiones a un servidor, quien le proporciona servicios y respuestas. El funcionamiento sería:

- El cliente envía una solicitud al servidor mediante su dirección IP y el puerto, que está reservado para un servicio en particular que se ejecuta en el servidor.
- El servidor recibe la solicitud y responde con la dirección IP del equipo cliente y su puerto.

En este trabajo podemos diferenciar 3 elementos del modelo cliente-servidor:

1. La base de datos: Será el lugar donde se encuentra toda la información de los pacientes y médicos.

2. Servidor Apache: Es el encargado de desplegar el servicio web REST, que recibirá las diferentes peticiones HTTP de los usuarios, algunas de ellas codificadas mediante JSON. En estas peticiones irá toda la información que el servidor pasará a procesar. El servicio web será el encargado de realizar las labores de comunicación e intercambio de información con la base de datos MySQL, mediantes sentencias SQL que estarán expresadas en PHP. Tras dicho intercambio con la base de datos, la información será devuelta al usuario en forma de respuestas HTTP, codificadas mediante JSON o en texto plano.
3. Usuario: Los diferentes usuarios que dispongan de la aplicación instalada en sus Smartphone o Tablet Android, podrán acceder a los diferentes servicios que ofrece dicha aplicación.

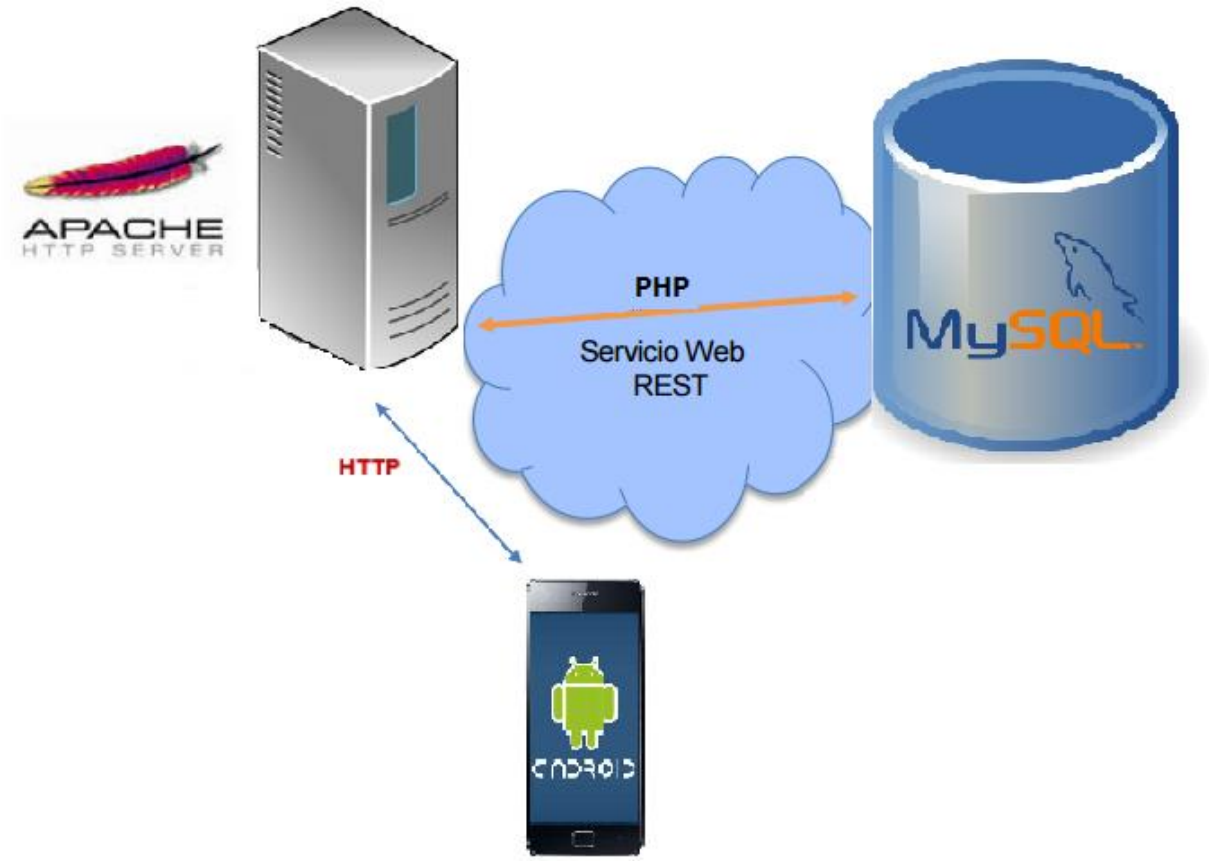


Figura 13. Arquitectura del servicio

3 RESULTADOS

El principal resultado de este proyecto es un servicio web para el almacenamiento y consulta de información de contexto desde los sensores integrados en un smartphone. Dicha información de contexto puede ser de utilidad para numerosos servicios de m-Salud y, como muestra, se han diseñado varios que hacen uso del servicio web para adaptarse al escenario del usuario. Se ha diseñado asimismo una aplicación que permite a un usuario (médico o cuidador) ver la información de monitorización de un paciente recogida por los sensores de su smartphone.

3.1 Análisis de sensores de Smartphone

A continuación se comentarán como están definidos los sensores en Android, qué tipo de datos obtienen y posibles usos:

- Sensor de temperatura: definido como TYPE_AMBIENT_TEMPERATURE, obtiene la temperatura en grados centígrados (ºC). Podría utilizarse para conocer si el usuario se encuentra en unos intervalos de temperatura aconsejables.
- Sensor de humedad: definido como TYPE_RELATIVE_HUMIDITY, obtiene el porcentaje de humedad relativa. Podría utilizarse para conocer si el usuario se encuentra en unos intervalos de humedad relativa (HR) aconsejables.
- Sensor de ritmo cardiaco: definido como TYPE_HEART_RATE, obtiene las pulsaciones por minuto. Podría utilizarse para controlar el pulso cardiaco del usuario aunque necesita de la interacción del usuario.
- Sensor de aceleración (acelerómetro): definido como TYPE_ACCELEROMETER. Obtiene tres medidas de la aceleración de cada uno de sus tres ejes en metros partidos por segundo al cuadrado (m/s^2). Estas medidas necesitan ser tratadas según el movimiento que se quiera detectar. Podría utilizarse para contar pasos o detectar caídas del usuario.
- Sensor giroscópico (giroscopio): definido como TYPE_GYROSCOPE. Obtiene tres medidas en radianes partido por segundo (rad/s) de la rotación alrededor de cada uno de los ejes. Podría utilizarse junto con el acelerómetro para detectar movimientos.
- Sensor de pasos: definido como TYPE_STEP_COUNTER o TYPE_STEP_DETECTOR. Utiliza el acelerómetro. Android distingue dos tipos de sensores aquí: uno que cuenta los pasos y otro que detecta pasos. Los dos saltan al detectar un paso, mientras que el primero obtiene el número de pasos que lleva desde que se inició el dispositivo, el otro es más útil para realizar una acción al detectar cada paso.
- Sensor de proximidad: definido como TYPE_PROXIMITY. Obtiene la distancia a la que se encuentra el usuario al dispositivo, generalmente solo dispone de dos medidas: cerca y lejos.
- Sensor de luz: definido como TYPE_LIGHT. Obtiene la medida en luxómetro (lux) de la luz del ambiente. Podría usarse para conocer si el usuario se encuentra en un entorno muy iluminado o poco.
- Sensor de presión: definido como TYPE_PRESSURE. Obtiene la presión del ambiente en hectopascales (hPa) o milibares (mbar).

Los elementos siguientes no están calificados dentro de los sensores, pero serán útiles para conocer mejor el contexto del usuario:

- Localización: obtiene la latitud, longitud y altitud del usuario y de ella se puede sacar la dirección, código postal, ciudad, estado y país.

- Audio: obtiene el modo de audio para las llamadas en el que se encuentra el dispositivo: normal (sonido), vibración o silencio.
- Pantalla: obtiene el estado de la pantalla: apagada o encendida.
- Llamada: obtiene el modo en el que se encuentra el dispositivo. llamada entrante, realizando llamada o sin actividad.

3.2 Diseño y despliegue de la base de datos

Una vez que hemos analizado los distintos elementos en los que se basará la información de contexto del usuario, a continuación procedemos al modelado de la base de datos que recogerá dicha información.

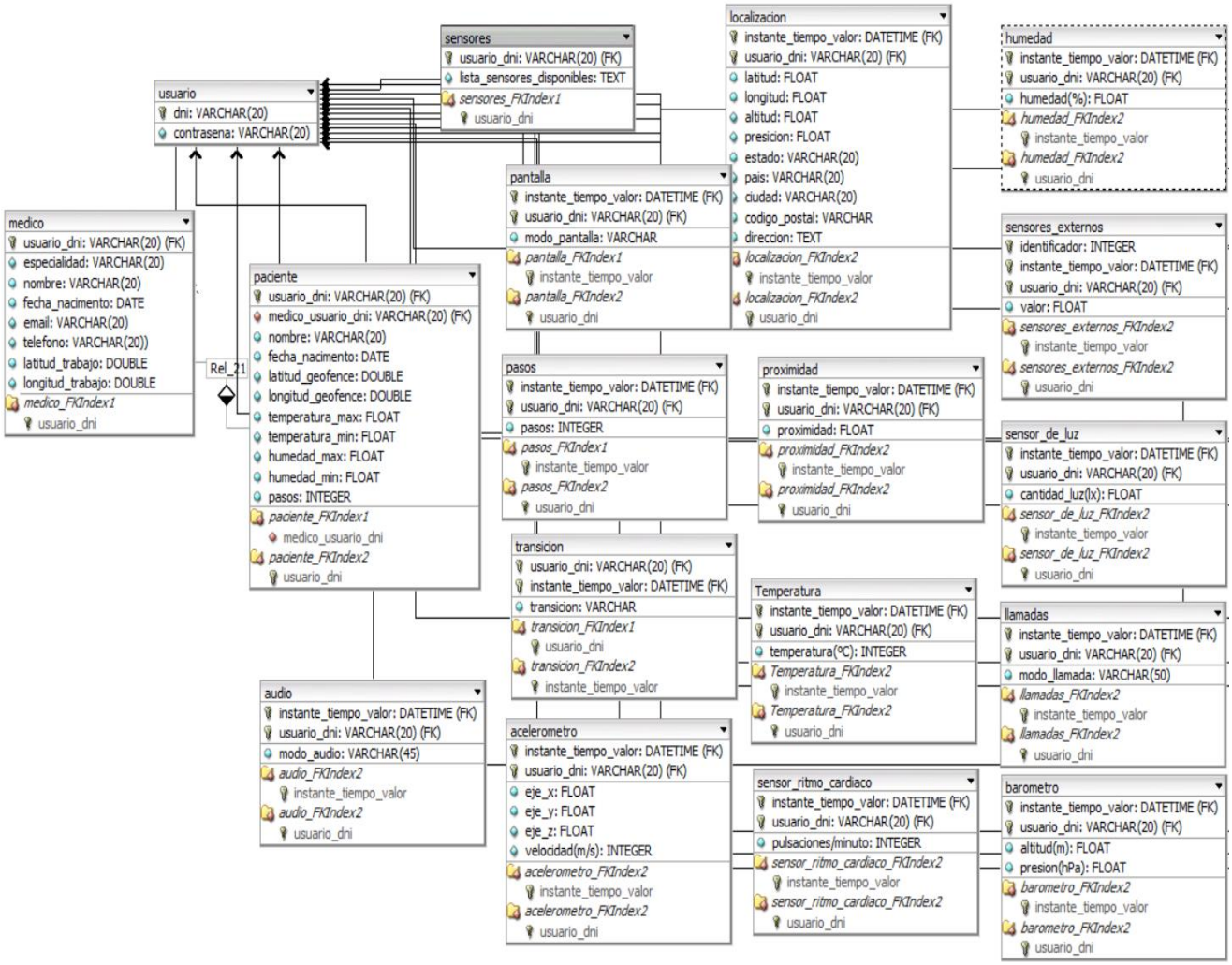


Figura 14. Diseño de la base de datos general

En este modelo se dispone de una entidad usuario de la cual dependen dos entidades débiles que son médico y paciente, que están relacionados de manera que cada paciente tiene un médico asignado. Con estas entidades se puede distinguir entre distintos usuarios y más allá, entre pacientes y médicos (Figura 15).

- Usuario: contiene el DNI y la contraseña con el que el usuario podrá iniciar sesión antes de acceder al resto del servicio web si es necesario.
- Paciente: contiene el DNI que hace referencia al usuario, el nombre del paciente, la fecha de nacimiento, el DNI del médico asignado a través de una relación con la entidad médico 1:n, es decir cada paciente tiene designado un médico, y cada médico monitoriza a varios pacientes. Además contiene la latitud y longitud para el servicio de monitorización de localización para pacientes con deterioro cognitivo, los umbrales de temperatura y humedad para el servicio de condiciones medioambientales del hogar para

pacientes de avanzada edad o recién operados y el umbral de números de pasos para el servicio de monitorización de la actividad para pacientes con obesidad o problemas cardiovasculares.

- Médico: contiene el DNI que hace referencia al usuario, la especialidad, el nombre, la fecha de nacimiento, el email y el teléfono, usados para enviar notificaciones cuando sea necesario, y la latitud y longitud del lugar de trabajo usado para conocer si el médico se encuentra en ese lugar a la hora de enviar notificaciones. Estas notificaciones serán explicadas en el apartado 3.4.

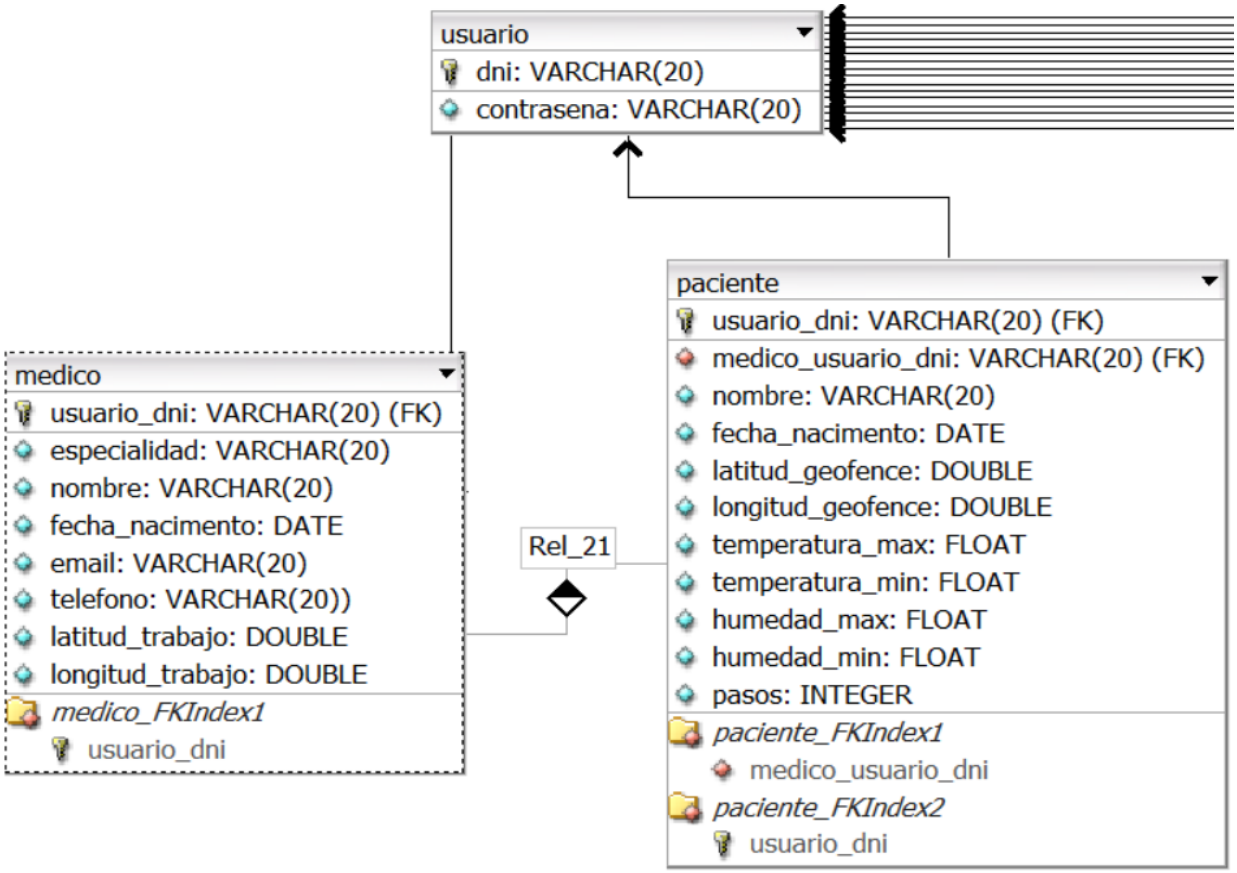


Figura 15. Diseño de la base de datos, tablas usuario, paciente y médico

Después se tienen todas las entidades de los sensores y los demás elementos para la información de contexto del usuario. Cada una está definida por una doble clave primaria compuesta de: una referencia al DNI del usuario y la fecha en la que se obtuvo la información (Figura 16).

- Temperatura: contiene la temperatura adquirida del sensor de temperatura en un instante de tiempo referente a un usuario.
- Sensor_ritmo_cardiaco: contiene las pulsaciones por minuto medidas con el sensor de ritmo cardiaco en un instante de tiempo referente a un usuario.
- Acelerómetro: contiene la aceleración de los tres ejes y la aceleración media mediadas con el acelerómetro en un instante de tiempo referente a un usuario.
- Proximidad: contiene la medida del sensor de proximidad en un instante de tiempo referente a un usuario.
- Humedad: contiene la humedad relativa medida a través del sensor en un instante de tiempo referente a un usuario.
- Sensor_de_luz: contiene la medida de la luz del ambiente medida con el sensor de luz en un instante de tiempo referente a un usuario.
- Presión: contiene la presión del ambiente medida con el sensor correspondiente en un instante de tiempo

referente a un usuario.

- Localización: contiene la latitud, longitud, altitud, dirección, código postal, ciudad, estado y país en un instante de tiempo referente a un usuario.
- Transición: contiene uno de los dos estados, entrando o saliendo, posibles del paciente respecto a un perímetro prefijado con la latitud y longitud de la tabla de paciente.
- Pasos: contiene el número de pasos medido en un instante de tiempo referente a un usuario.
- Audio: contiene el modo de audio en el que se encuentra el dispositivo, normal, vibración o silencio.
- Pantalla: contiene el estado de la pantalla en un instante de tiempo referente a un usuario.
- Llamada: contiene el modo de llamada en el que se encuentra el dispositivo en un instante de tiempo referente a un usuario.
- Sensores: contiene los sensores disponibles en el móvil de cada usuario.

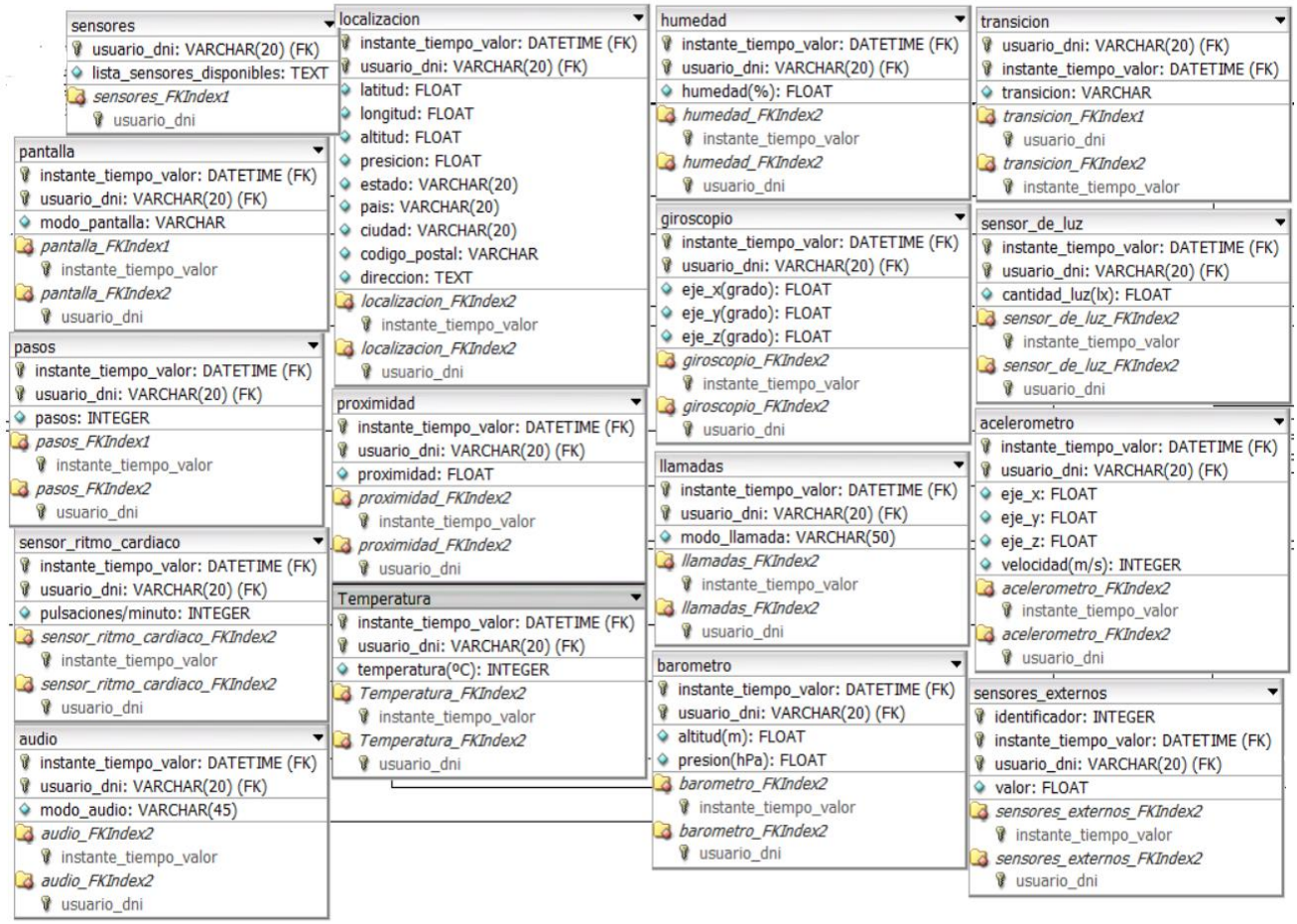


Figura 16. Diseña de la base de datos, tablas de información de contexto

3.3 Servicio web para el acceso a información de contexto

Como se comentó anteriormente la decisión tomada para la creación del servicio eb fue que estuviera basado en REST. Para esto se necesita en funcionamiento el servidor Apache proporcionado por la aplicación XAMPP y se deben crear los métodos necesarios a los que accederán las peticiones HTTP.

Se comenzará explicando las 4 etapas de la comunicación con el servicio web (Figura 17):

1. El cliente envía un mensaje HTTP hacia el servicio web, indicando la URI donde se encuentra el recurso que va a consumir. La petición será del tipo POST. Según el recurso a utilizar el usuario deberá mandar

uno o más parámetros, estos parámetros se incluirán en el cuerpo del mensaje al ser la petición POST.

2. El servicio web identifica por medio de la URI el recurso que se va a utilizar, además de los parámetros.
3. El propio recurso web, si el método es de petición y los parámetros son correctos, se pondrá en contacto con la base de datos y obtendrá la información de ella. Tras esto construye la respuesta en formato JSON.
4. El servicio web envía la repuesta HTTP al cliente, que se encargará de procesarla adecuadamente y de la que obtendrá un array tipo JSON, en el que contendrá bien las filas obtenidas de la base de datos si el recurso era para la consulta de datos o un mensaje de error o éxito.

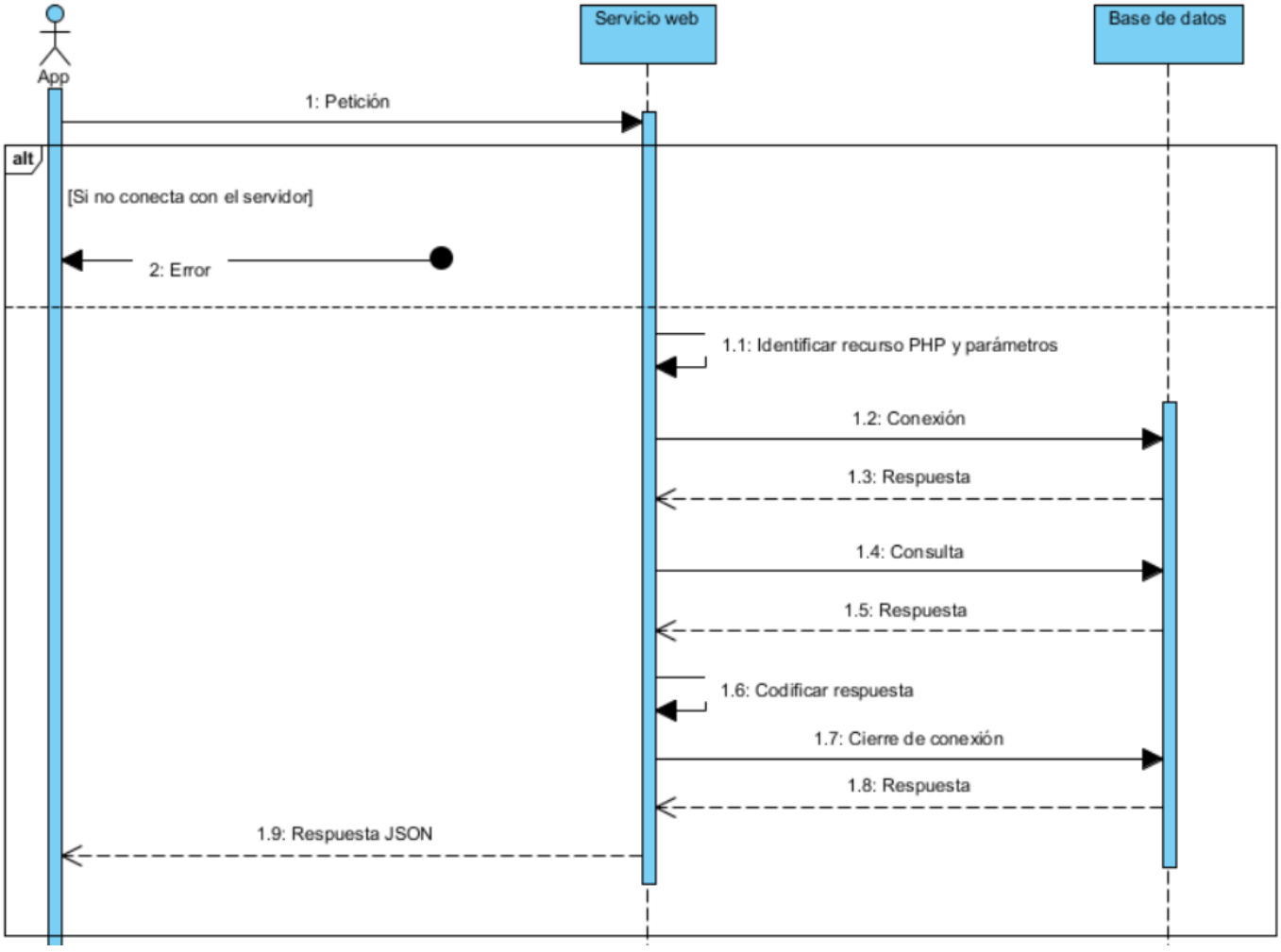


Figura 17. Diagrama de comunicación entra la aplicación y el servidor

Para realizar las peticiones HTTP usaremos unas URI más amigables que redireccionarán al método correspondiente. A continuación se explicarán los distintos recursos, junto con sus URI correspondientes y los parámetros necesarios de cada uno. Los métodos podemos clasificarlos en cuatro grupos: los que insertan (insertan una o más filas en la base de datos), los que modifican (actualizan una o más filas de la base de datos), los que muestran (muestran una o más filas de la base de datos) y los que comprueban (comprueban que existe un usuario o si este es paciente o médico). Estos métodos se encuentran en el Anexo A.

Tabla 1. Relación entre URI, parámetros y métodos

URI	Parámetros	Método
/insertarAcelerometro	fecha, eje_x, eje_y, eje_z, aceleracion y dni	**insertarAcelerometro.php**
/insertarAudio	fecha, modo_audio, dni	**insertarAudio.php**
/insertarPantalla	fecha, modo_pantalla, dni	**insertarPantalla.php**
/insertarLlamada	fecha, modo_audio, dni	**insertarLlamada.php**
/insertarHumedad	fecha, humedad, dni	**insertarHumedad.php**
/insertarLocalizacion	fecha, latitud, longitud, altitud, precisionn, pais, estado, ciudad, codigo_postal, direccion, dni	**insertarLocalizacion.php**
/insertarLuz	fecha, luz, dni	**insertarLuz.php**
/insertarLocTrabajo	longitud, latitud, dni	**insertarLocTrabajo.php**
/insertarPasos	fecha, pasos, dni	**insertarPasos.php**
/insertarPresion	fecha, presion, dni	**insertarPresion.php**
/insertarProximidad	fecha, aproximacion, dni	**insertarProximidad.php**
/insertarTemperatura	fecha, temperatura, dni	**insertarTemperatura.php**
/insertarTransicion	fecha, transicion, dni	**insertarTransicion.php**
/insertarUsuario	dni, contrasena, lista_sensores_disponibles, nombre, fecha_nacimiento, [medico_dni], [latitud_geofence], [longitud_geofence], [especialidad], [email],[telefono]	**insertarUsuario.php**
/modificarMedico	email, dni	**modificarMedico.php**
/modificarPaciente	dni, [temperatura_max], [temperatura_min], [humedad_max], [humedad_min], [pasos]	**modificarPaciente.php**
/mostrarAudio	dni	**mostrarAudio.php**
/mostrarPantalla	dni	**mostrarPantalla.php**
/mostrarLlamada	dni	**mostrarLlamada.php**
/mostrarProximidad	dni	**mostrarProximidad.php**
/mostrarDatosFecha	[fecha], dni, dato	**mostrarDatosFecha.php**

/mostrarEmailTelefono	dni	**mostrarEmailTelefono.php**
/mostrarUltimaLocalizacion	dni	**mostrarUltimaLocalizacion.php**
/mostrarNombresPacientes	dni	**mostrarNombresPacientes.php**
/mostrarLocGeofence	dni	**mostrarLocGeofence.php**
/mostrarLocTrabajo	dni	**mostrarLocTrabajo.php**
/mostrarHumedad	dni	**mostrarHumedad.php**
/mostrarTemperatura	dni	**mostrarTemperatura.php**
/mostrarPasos	dni	**mostrarPasos.php**
/comprobarPacientes	dni	**comprobarPacientes.php**
/comprobarUsuario	dni	**comprobarUsuario.php**

Todos los métodos hacen uso de otro método llamado *conection.php*, que contiene los datos de acceso a la base de datos: usuario, contraseña, nombre de la base de datos y servidor donde se encuentra. También contiene la clase *Respuesta* necesaria para crear la respuesta de la petición HTTP en formato JSON.

Los métodos que insertan filas, modifican o comprueban, si todo va correctamente responden con un mensaje HTTP 200 (OK) indicando el éxito usando la clase *Respuesta*. Los métodos que muestran datos responden con un array de tantos objetos JSON como filas devuelvan. Si se produce algún error, el servicio devolverá un mensaje HTTP 422 (*Unprocessable Entity*), en caso que algún parámetro este erróneo, un mensaje HTTP 400 (*Bad Request*), si el tipo de petición no es el indicado y un mensaje HTTP 404 (*Not Found*), si no encuentra el recurso especificado.

Para conseguir el redireccionamiento desde la URI al método correspondiente se modifica el archivo httpd.conf de Apache mediante la directiva Alias como se ve en la Figura 18.

```
Alias /comprobarUsuario C:/xampp/htdocs/tfg/comprobarUsuario.php
```

Figura 18. Redireccionamiento hacia el método mostrarUsuario.php

Un ejemplo del uso del servicio web con el método *comprobarUsuario.php* (el cual comprueba que exista un usuario con el DNI y la contraseña pasados por parámetro) se describe a continuación. Si el usuario existe, el servicio nos responde con el mensaje JSON: Usuario existe (Figura 19).

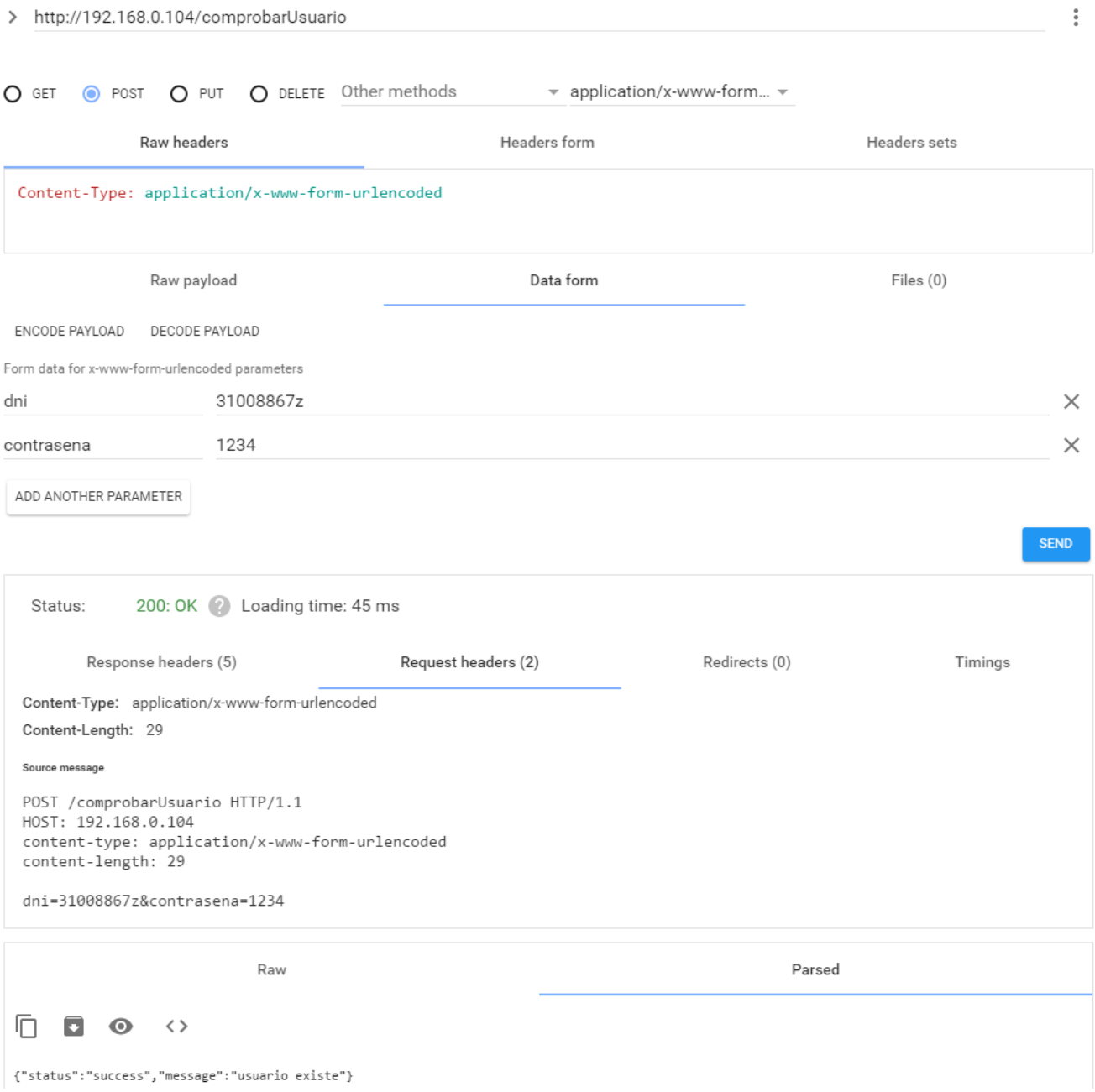


Figura 19. Prueba realizada con el software Advanced REST Client de éxito

3.4 Aplicación Android de ejemplo

Como ejemplo se propone una aplicación tanto para pacientes como para médicos que haga uso del servicio. Con esta aplicación se podrá monitorizar el contexto de los pacientes y los médicos, haciendo posible tener un control más exhaustivo de pacientes en su hogar y que se puedan enviar avisos ante posibles incidencias. La monitorización constará de una recogida de información que será insertada en la base de datos a cada minuto, o cuando esta cambie según el tipo de información.

Se monitorizará:

- La localización, cada minuto.
- La temperatura y la humedad, solo en el caso del paciente y cada minuto.
- El número de pasos, solo en el caso del paciente, cada minuto.
- La luz, cada minuto.
- La presión, solo en el caso del paciente, cada minuto.
- La proximidad, cada vez que cambie.
- Audio, llamadas y pantalla, cada vez que cambien.

La información de los pacientes se puede obtener de modo pull, el médico la solicita en la misma aplicación, o push, el médico es avisado de algún incidente.

Pueden detectarse las siguientes incidencias:

- La temperatura de ambiente es menor o mayor que los umbrales puestos por el médico, que pueden estar especificados en la base de datos del servicio, en la tabla del paciente.
- La humedad relativa de ambiente supera o es menor que los umbrales puestos por el médico, que pueden estar especificados en la base de datos del servicio, en la tabla del paciente.
- El paciente llega a un número de pasos definido por el médico, en un periodo de 24 horas, que pueden estar especificados en la base de datos del servicio, en la tabla del paciente.
- Se sale o se entra en un perímetro prefijado en caso de pacientes que no deben alejarse una distancia determinada de su hogar. Este perímetro estará prefijado en la base de datos, en la tabla del paciente.

Se usará la monitorización del médico para conocer su información de contexto y en función de esto decidir la forma en la que se le notificará, pudiéndose dar los siguientes casos:

- Si está fuera del hospital o zona de trabajo con el móvil apagado o más de una hora sin monitorización, enviar un email.
- Si está fuera del hospital o zona de trabajo y realizando una llamada: recibir SMS.
- Si está fuera del hospital o zona de trabajo con el móvil en silencio: recibir un SMS con la información resumida.
- Si está fuera del hospital o zona de trabajo con el móvil con sonido o vibración: llamarle y avisarle con una locución.
- Si está en el hospital o zona de trabajo y la pantalla encendida: recibir SMS.
- En el hospital o zona de trabajo con pantalla apagada: enviarle un email.

Esta zona de trabajo del médico será especificada al crear el usuario y guardada en la base de datos en la tabla del médico.

3.4.1 Diseño de la aplicación

Cuando un usuario inicia la aplicación puede iniciar sesión o crear un nuevo usuario. Al iniciar sesión se distingue entre paciente o médico (Figura 20).

- Paciente: accederá a la pantalla de monitorización con dos botones para iniciar y finalizar el control.
- Médico: primero accede a la pantalla de la lista de pacientes, donde se mostrarán todos sus pacientes asignados, al pulsar sobre uno de ellos se muestra un formulario para solicitar información de la monitorización del paciente.

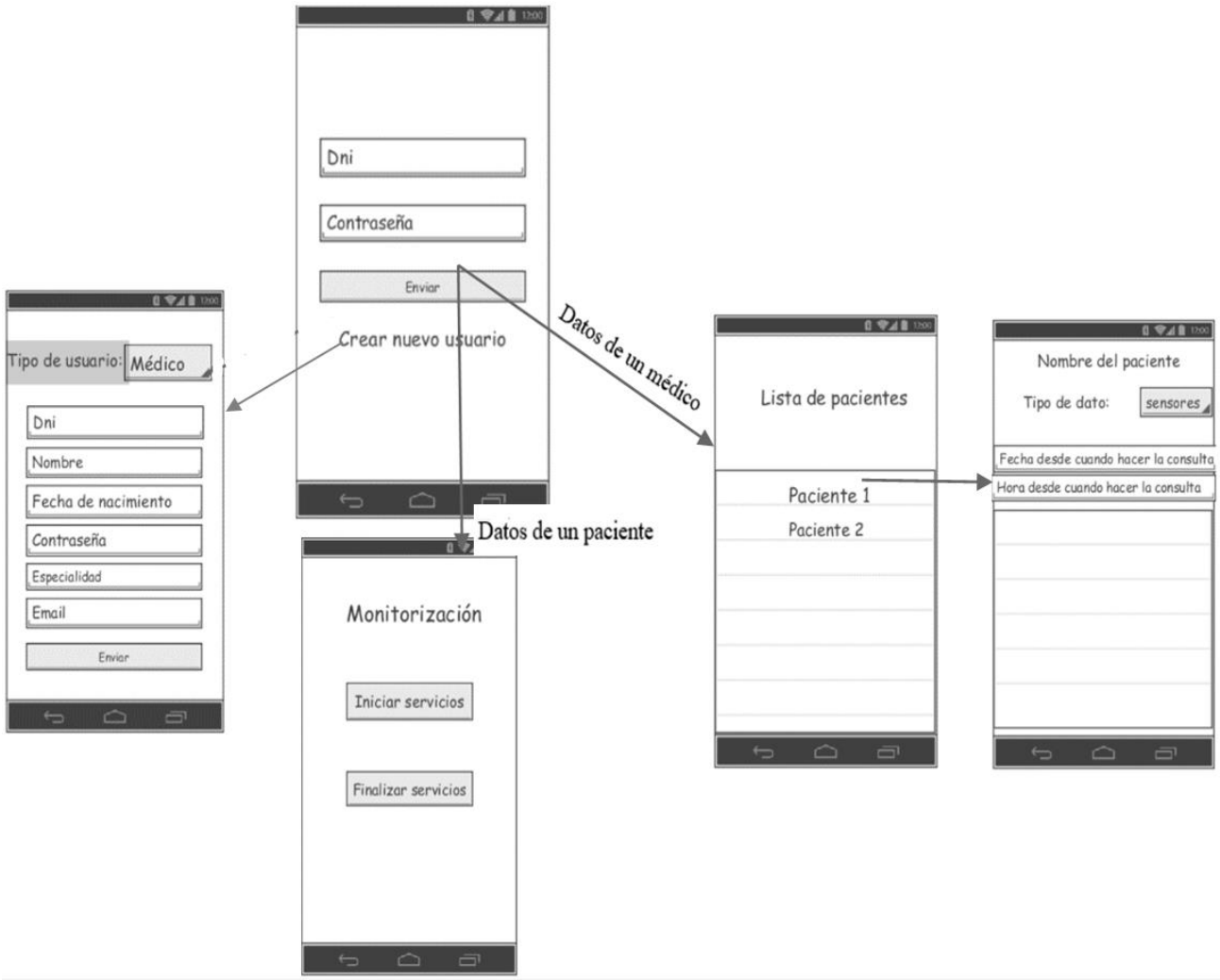


Figura 20. Diseño de las principales interfaces de usuario de la app

El médico cuenta en cada una de sus dos pantallas con un menú con las siguientes opciones (Figuras 21 y 22):

- Cerrar sesión, vuelve a la pantalla de inicio, deteniendo la monitorización si esta estaba activa.
- Configuración, le permite configurar el email al que le enviaran las notificaciones cuando el usuario está en la pantalla del listado de pacientes. Si se encuentra dentro de la información de un paciente, el ítem Configuración le llevará a configurar los umbrales de las incidencias.
- Pacientes, le lleva de vuelta a la pantalla de la lista de pacientes.

- Monitorización, se accede a la pantalla de monitorización del médico.

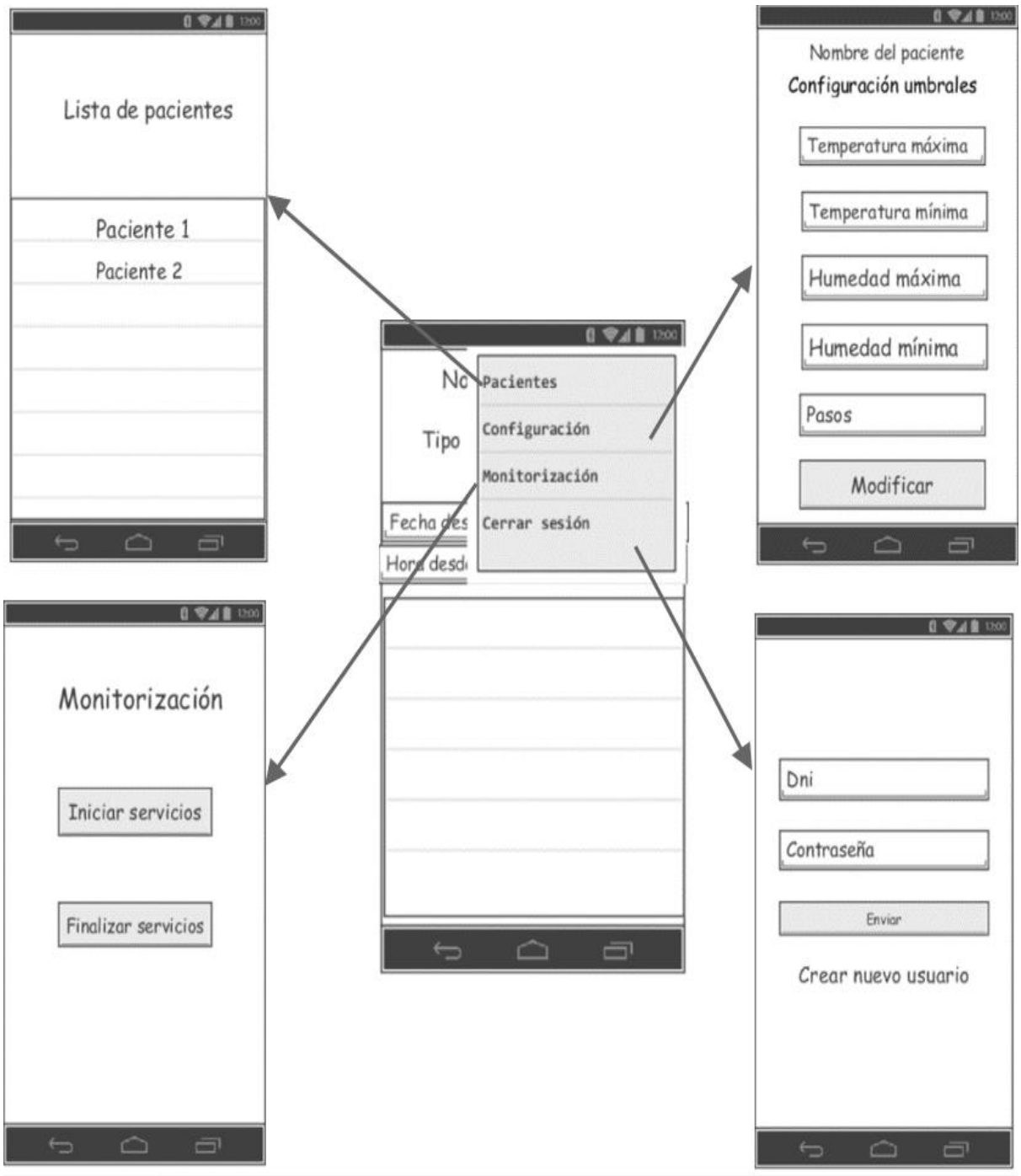


Figura 21. Diseño del menú del médico cuando está dentro del perfil de un paciente

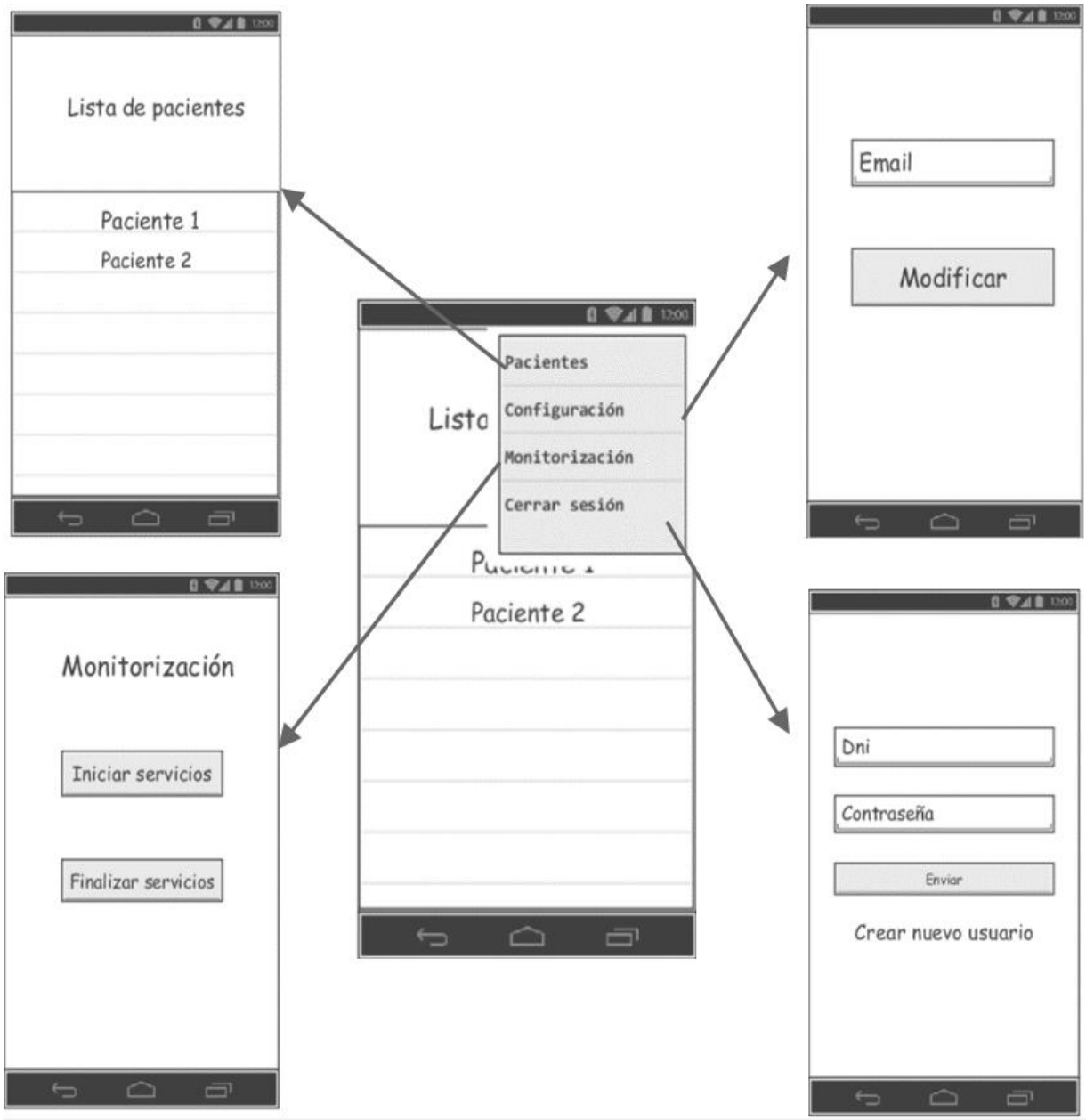


Figura 22. Diseño del menú del médico cuando está en el listado de pacientes

El menú del paciente consiste simplemente en la opción cerrar sesión, la cual te permite volver a la pantalla principal, deteniendo la monitorización si esta estaba activa (Figura 23).

Las Figuras 24 y 25 explican en forma de diagrama las actividades de un paciente y un médico en la aplicación.

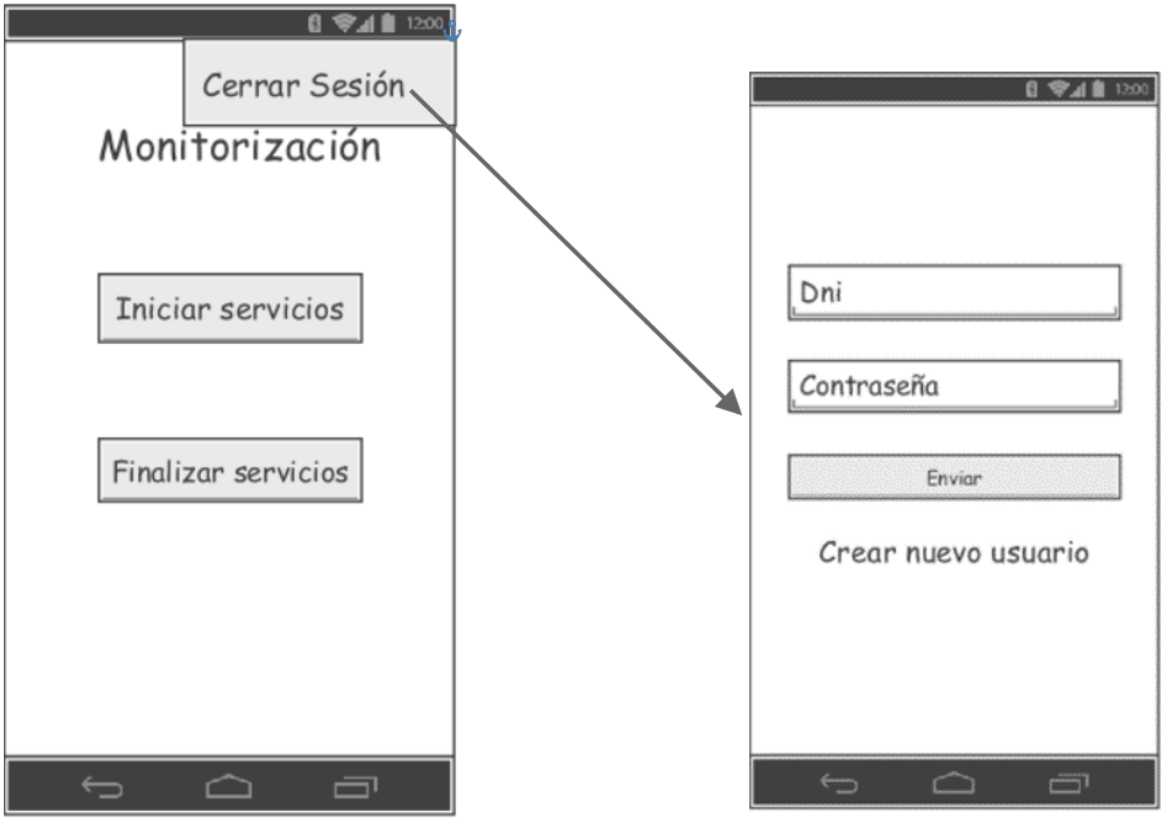


Figura 23. Diseño del menú del paciente

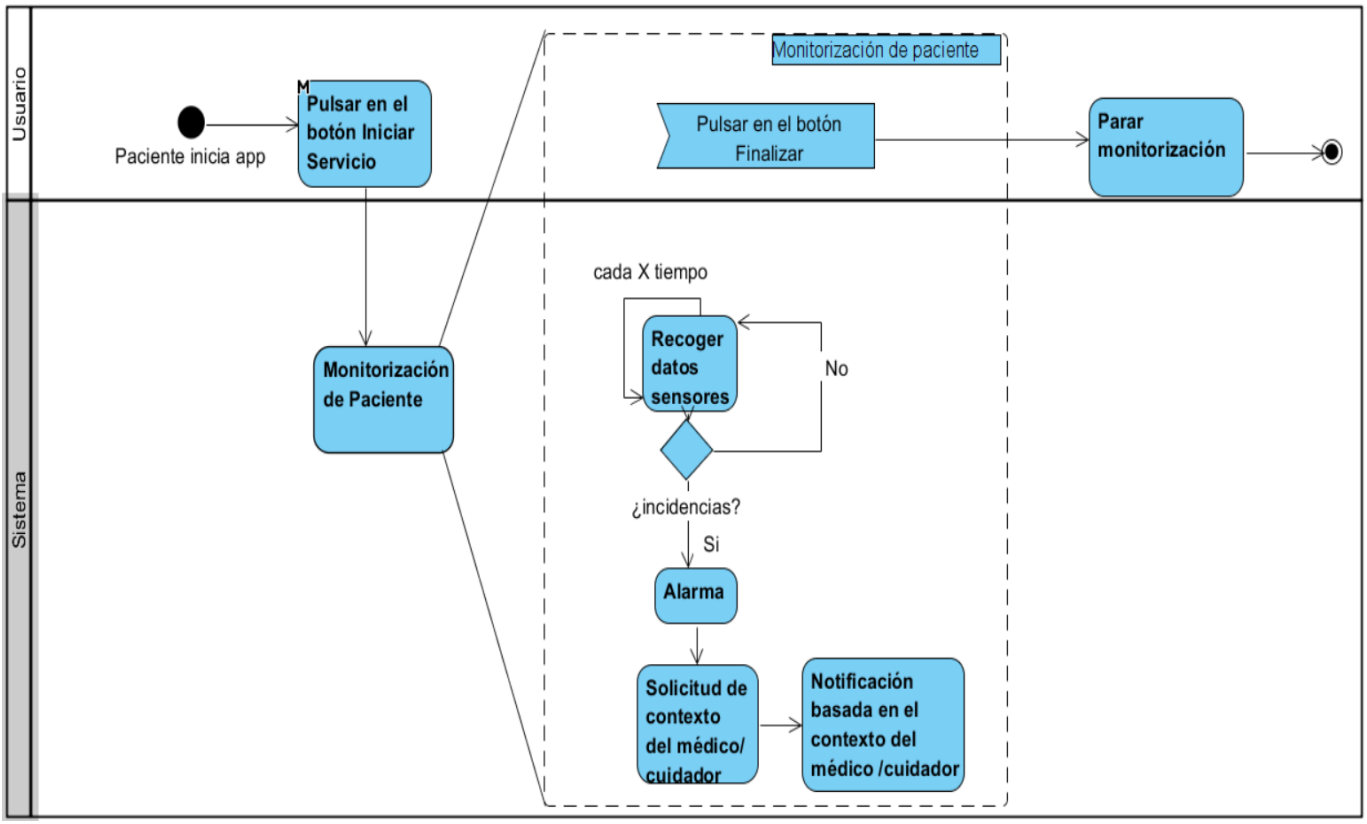


Figura 24. Diagrama de actividad resumen del paciente

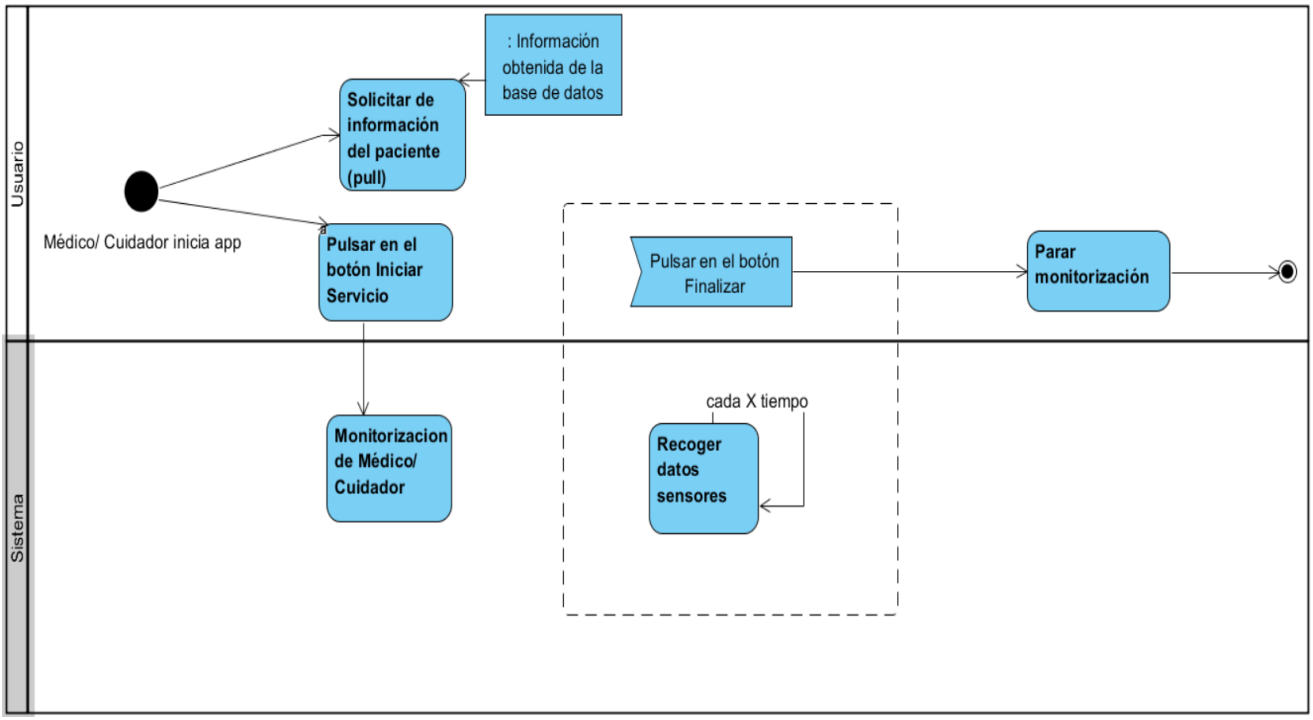


Figura 25. Diagrama de actividad resumen del médico

3.4.2 Interacción con el servicio web

Es necesario tener claro el modo en el que la aplicación se comunicará con el servicio web. Volley [22] es una librería desarrollada por Google para optimizar el envío de peticiones HTTP desde las aplicaciones Android hacia servicios externos. Este componente actúa como una interfaz de alto nivel, liberando al programador de la administración de hilos y procesos tediosos de parsing, para permitir publicar fácilmente resultados en el hilo principal, Volley ofrece solicitudes de conexiones de red automáticamente, múltiples conexiones de red simultáneas, soporte para priorización de las peticiones, cancelación de peticiones e implementación de caché en disco y memoria.

Esta librería tiene limitaciones con la descarga de información demasiada extensa, ya que su operación se basa en salvaguardas en cache, lo que haría lento el proceso con datos voluminosos. Por ahora basta esta librería al usar datos pequeños, habría que probar si en una base de datos más extensa seguiría funcionando correctamente.

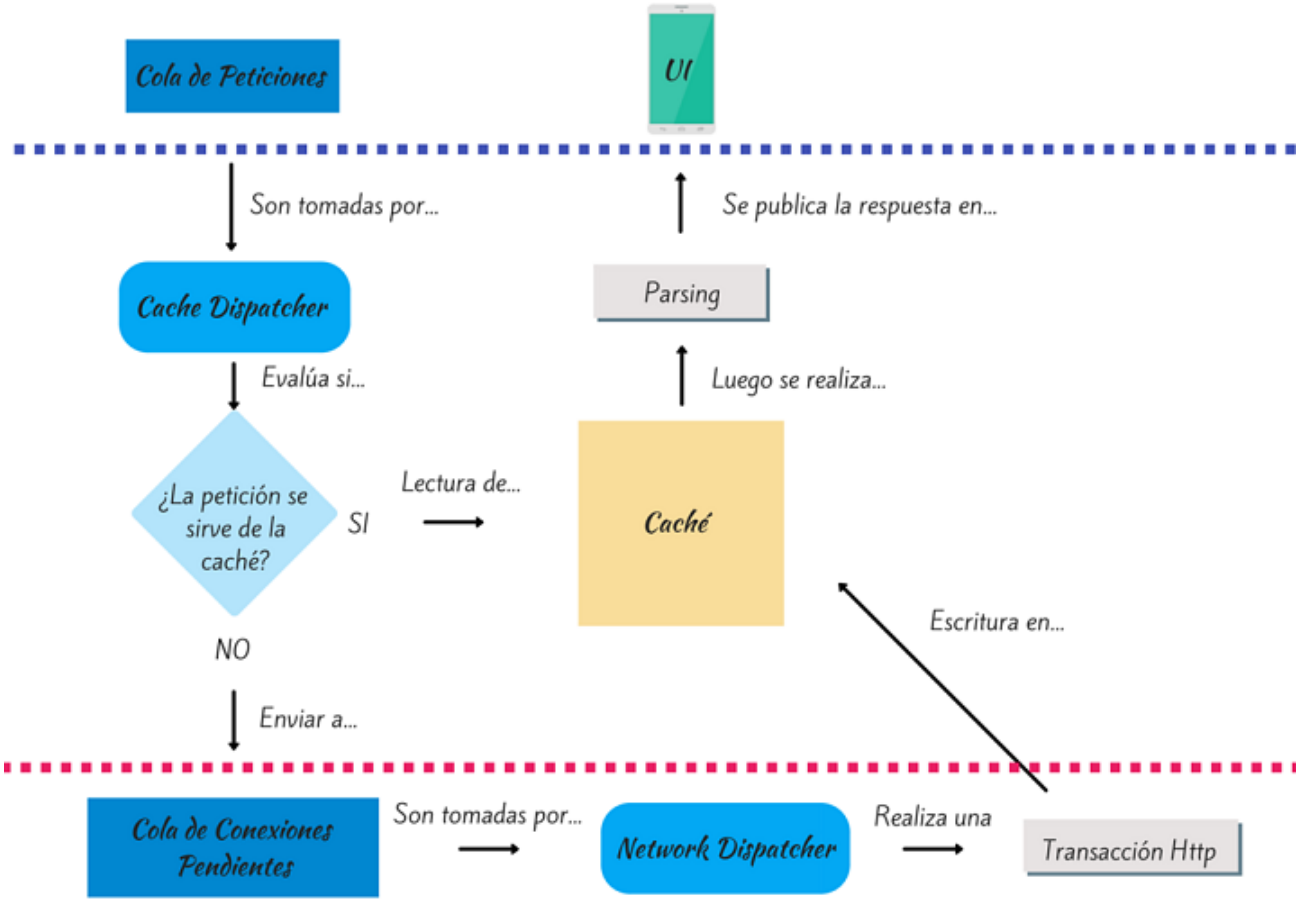


Figura 26. Funcionamiento petición Volley

```
private RequestQueue requestQueue;
```

Figura 27. Declaración de la cola de peticiones

```
requestQueue  = Volley.newRequestQueue(getApplicationContext());
```

Figura 28. Creación de la cola de peticiones

Con la librería Volley primero se crea una cola de peticiones (Figura 28). Con el método StringRequest se crea una petición GET o POST. Se usará POST. Se indican:

- La URL con la IP del servicio más la URI que hace referencia al recurso que se necesita
- La acción llevada a cabo si se produce la petición sin errores.
- La acción en caso de que se produzcan errores.
- Los parámetros que le pasamos.

Tras esto se añaden a la cola de peticiones para que sea atendida:

```
StringRequest request = new StringRequest(Request.Method.POST, "http://"+ip+"/comprobarUsuario", new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        result=response;
        comprobarUsuario();
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        Log.d(TAG, "Error Volley: " + error.getMessage());
        Toast.makeText(getApplicationContext(),
                "Problema conectando con el servidor.", Toast.LENGTH_LONG).show();
    }
}){
    @Override
    public Map<String, String> getParams() throws AuthFailureError {
        Map<String, String> parameters = new HashMap<~>();
        parameters.put("dni", dni);
        parameters.put("contrasena", pass);
        return parameters;
    }
};

requestQueue.add(request);
```

Figura 29. Añadir petición a la cola de peticiones

Faltará tratar los datos del String recibido en formato JSON. Con el método Split podemos separar esta cadena y obtener los datos necesarios.

A continuación se explicarán las distintas clases creadas para el funcionamiento de la aplicación. Se dividen en dos grupos: los Activity que son los encargados de permitir que el usuario interactúe con la aplicación y los Service, los cuales se ejecutan en un segundo plano mientras está la monitorización en marcha.

3.4.3 ServidorActivity - Configurar servidor

Figura 30. Layout de ServidorActivity

La primera vez que se inicia la aplicación será necesario añadir la IP del servidor, ya que según en la red Wi-Fi en la se encuentre esta cambiará (Figura 32).

Cuando el usuario escribe la IP (hay una por defecto que es la utilizada en el lugar de pruebas y sirve como modelo) y pulsa en conectar, se inicia el método *conexionServidor(View view)* el cual guarda la dirección IP mediante la clase *SharedPreferences*. Esta clase permite recordar un valor de una manera limpia y muy sencilla.

Lo realmente interesante es que este valor permanece grabado aunque cerremos nuestra aplicación o reiniciemos el móvil.

```
settings = getSharedPreferences("perfil", MODE_PRIVATE);

SharedPreferences.Editor editor = settings.edit();
editor.putString("ip", ip);
editor.apply();
```

Figura 31. Modo de usar SharedPreferences

Antes de probar si se conecta correctamente al servidor, se comprueba si se tiene el Wi-Fi conectado, ya que si no daría error.

Después se realiza una petición al servicio de prueba con el método *conection.php*. Si se conecta correctamente iniciamos la actividad *MainActivity*.

Para las siguientes veces que se inicie la app esta comprueba si la IP ya ha sido asignada para iniciar directamente la actividad *MainActivity*.

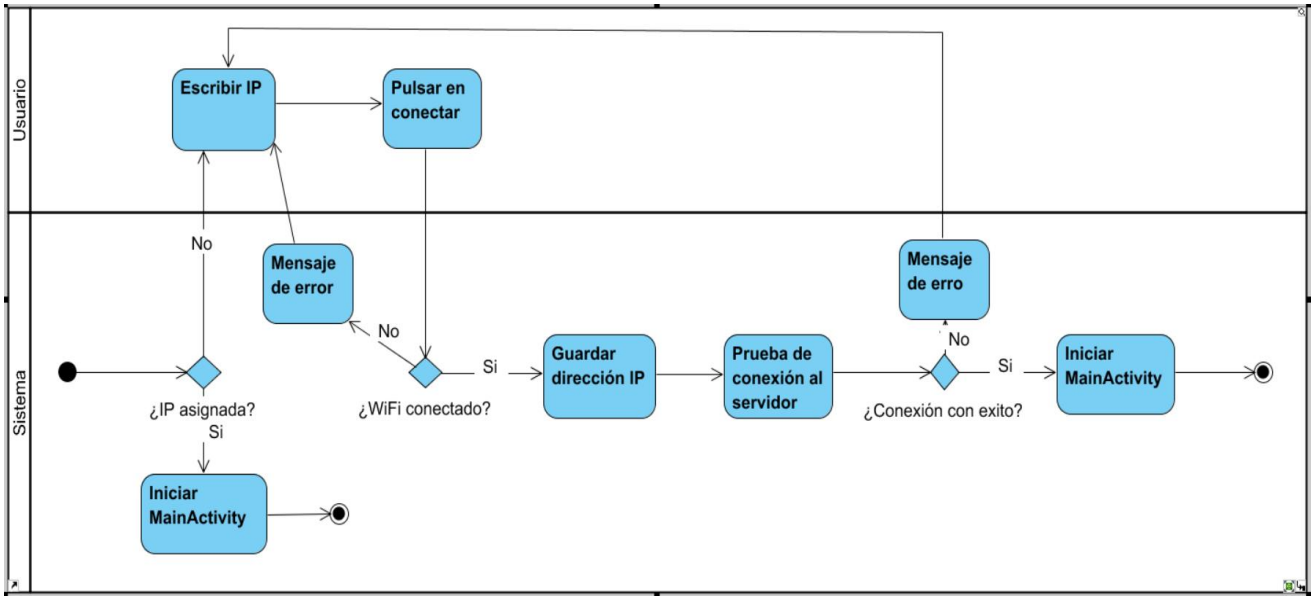


Figura 32. Diagrama de actividad de ServidorActivity

3.4.4 MainActivity - Inicio.

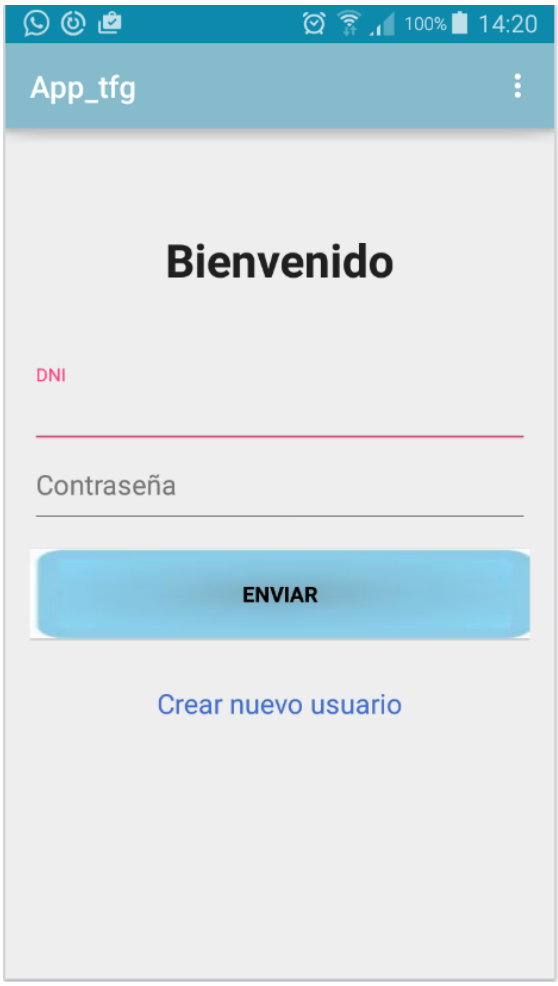


Figura 33. Layout de MainActivity

Esta clase es el primera Activity que vemos al iniciar la aplicación una vez que hayamos configurado la IP. Le pide al usuario el DNI y la contraseña para iniciar sesión (Figura 34).

Una vez que se pulsa el botón enviar, se llama al método *enviarDatos(View view)*, el cual comprueba que ninguno de los campos esté vacío y que el DNI tenga un formato correcto (8 número y una letra). Si está todo correcto hasta aquí, realiza una petición al servicio web con la URI /*comprobarUsuario*, suponiendo que conecta sin problema. El servicio comprobará si existe ese usuario con el DNI y la contraseña introducidos, devolviendo a la aplicación el mensaje de usuario existe o usuario no existe.

Al finalizar la petición se llama al método *comprobarUsuario()* que trata los datos devueltos por la petición. Si el usuario no existe o la contraseña es incorrecta se informa al usuario con un Toast. En caso de que exista se llama al método *comprobarPaciente()* para decidir si este usuario es un médico o un paciente. Se vuelve a hacer una petición al servicio con el método con la URI /*mostrarPacientes*. Devolviendo a la aplicación el mensaje de usuario existe o usuario no existe.

Antes de finalizar se llama al método *siguienteAct()*, que inicia la clase *ListaPacientesMedicoActivity* o *MonitorizacionPacienteActivity*, según si el usuario es un médico o un paciente. Pasándole como parámetro el DNI del usuario.

Si el usuario pulsa sobre *Crear un nuevo usuario*, se llama al método *CrearCuenta(View view)* y se inicia la clase *CrearCuentaActivity*.

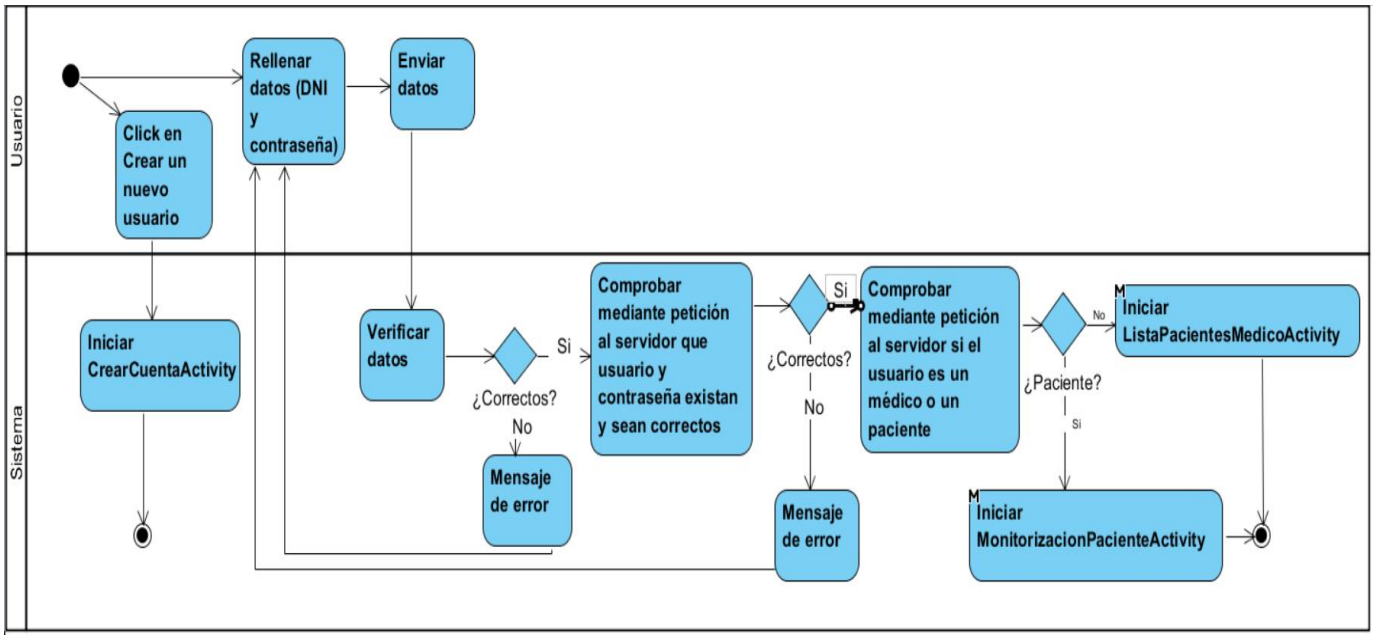


Figura 34. Diagrama de actividad de MainActivity

3.4.4.1 Menú de Inicio

En MainActivity disponemos de un menú con el ítem *Configuración*, el cual nos lleva a la clase *ConfiguracionServidorActivity* idéntica a *ServidorActivity*, salvo que no comprueba si ya hay una IP asignada para poder cambiarla.

3.4.5 CrearCuentaActivity

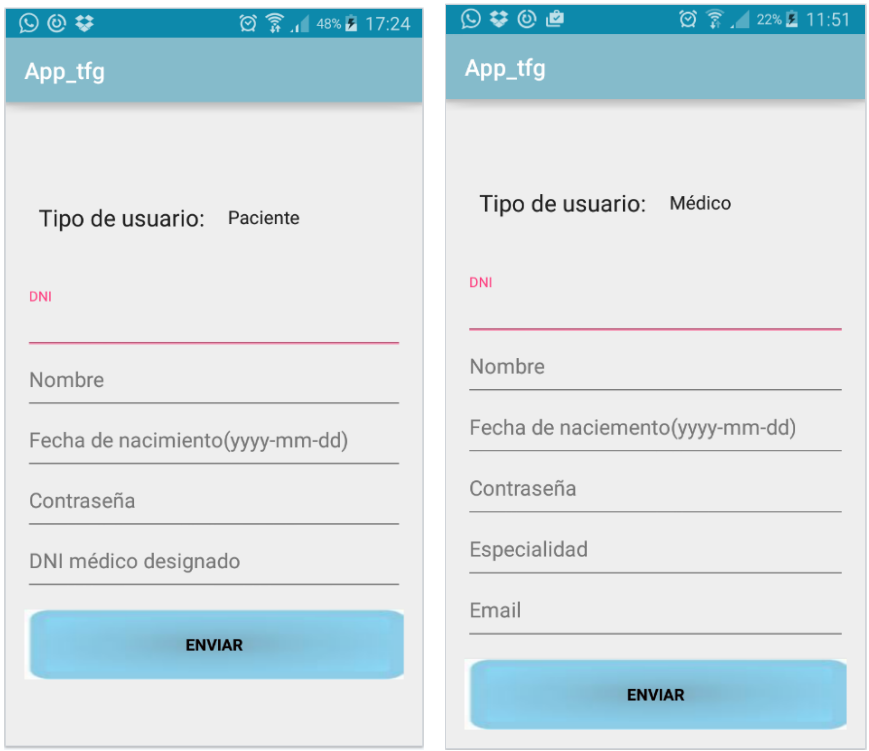


Figura 35. Layout de CrearCuentaActivity

Esta clase permite al usuario crear una cuenta, en el formulario hay datos fijos, el DNI, nombre, fecha de nacimiento, contraseña y tipo de usuario, según este último aparecen unos campos u otros. Estos campos están en modo de visibilidad GONE, es decir no se ven ni ocupan el espacio correspondiente, pero según lo seleccionado en el spinner se elige cuáles pasan a VISIBLE y cuales a GONE. Por defecto está seleccionado médico.

En cuanto se inicia la clase el sistema accede tanto a los sensores disponibles en el dispositivo móvil como al número de teléfono del dispositivo. Estos datos son también insertados en la base de datos al crearse el usuario.

```
SensorManager sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
List<Sensor> listaSensores = sensorManager.getSensorList(Sensor.TYPE_ALL);
int i=1;
for(Sensor sensor: listaSensores) {
    sensores=sensores+Integer.toString(i)+") "+sensor.getName() + "      ";
    i=i+1;
}

TelephonyManager mTelephonyManager;
mTelephonyManager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
telefono= mTelephonyManager.getLine1Number();
```

Figura 36. Modo de obtener los sensores disponibles y el número de teléfono

Una vez que se envían los datos se llama al método *enviarDatosC(View view)* el cual comprueba que ningún dato visible está vacío, que el DNI sea adecuado (8 números y una letra), el formato de la fecha sea el indicado (yyyy-mm-dd) y el email contenga un '.' y una @. Si todo es correcto, se realiza una petición al servicio web con la URI */insertarUsuario*. Suponiendo que se realiza correctamente y no existiera el usuario se procede a iniciar sesión si el usuario es un paciente yendo a la clase *MonitorizacionPacienteActivity* o iniciando la clase *MapsActivity* si el usuario es un médico para configurar el lugar de trabajo, pasándole como parámetro a ambos el DNI del usuario. En caso de que este ya existiera se avisa al usuario con un Toast.

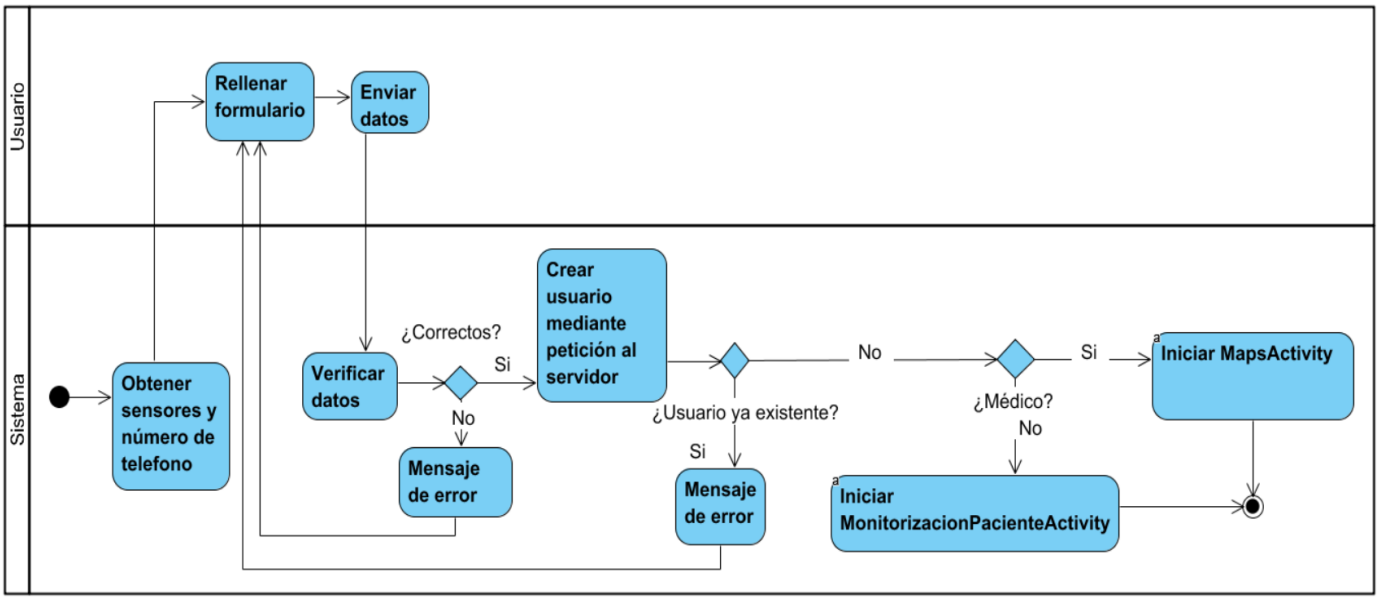


Figura 37. Diagrama de actividad de CrearCuentaActivity

3.4.6 MonitorizacionPacienteActivity

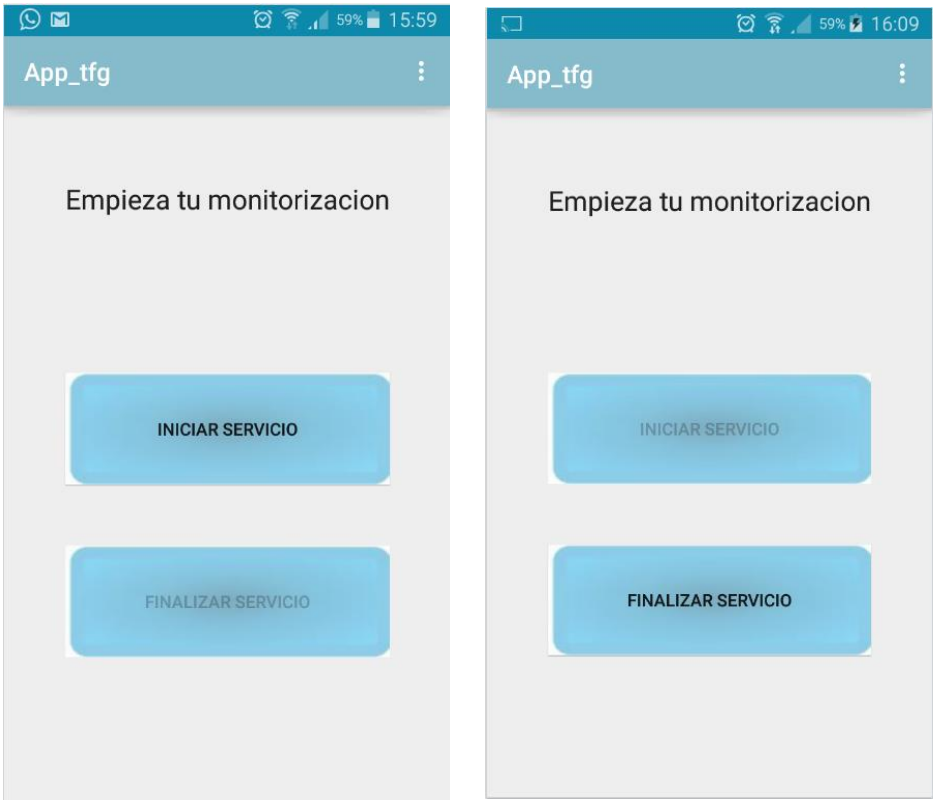


Figura 38. Layout de MonitorizacionPacienteActivity

Esta clase dispone de dos botones (Figura 38) y solo uno de ellos estará activo, según si los servicios están corriendo o no. Si alguno de los servicios utilizados para monitorizar al médico está activo, se desactiva el botón para iniciar la monitorización y se activa el botón de finalizar, y viceversa.

Al pulsar sobre el botón *Iniciar servicio* se ejecuta el método *Start_Servicios(View view)* para iniciar en segundo plano cada uno de los servicios necesarios para la monitorización introduciendo como parámetro el DNI del médico. Al iniciar la monitorización se puede seguir navegando por la aplicación o incluso salirse de ella, ya que los servicios seguirán ejecutando en segundo plano hasta que se pulse el botón de Finalizar servicio o se cierre la sesión desde el menú.

Al pulsar sobre *Finalizar servicio* se ejecuta *Stop_Servicios(View view)* el cual para cada uno de los servicios.

Al empezar la monitorización se crea una notificación en el dispositivo del usuario como recordatorio y para poder volver a la pantalla de la monitorización al pulsar sobre ella (Figura 39).

Figura 39. Notificación de monitorización

Además de cada uno de los servicios necesarios para la monitorización también se añade un Geofence, facilitado por los servicios de Google.

3.4.6.1 Google Location Services API

Para el desarrollo de aplicaciones de localización pueden elegirse dos Application Programming Interface (API):

* android.Location [23]

* Google Location Services API [24]

Se optó por el uso de la segunda, ya que de acuerdo a la documentación de Android es preferible [25]. Esta API nos proporciona una precisión mayor y un menor consumo de batería que la API android.Location.

El termino Geofencing se refiere a la combinación entre la localización actual del usuario y el conocimiento de la proximidad del usuario a una localización de interés. La localización de interés se especifica con latitud y longitud, el área de proximidad con un radio. Latitud, longitud y radio definen un geofencing, creando un área circular alrededor de la localización de interés.

Con la localización actual del usuario se determinaría si se entra o se sale de esta área.

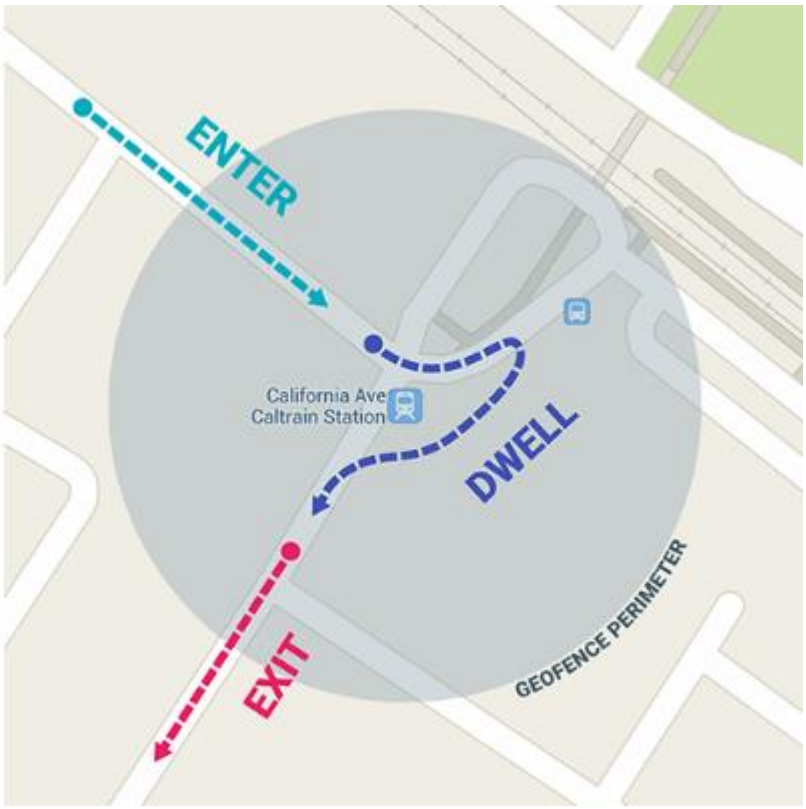


Figura 40. Transiciones de Geofencing

El Geofencing será útil para controlar pacientes con deterioro cognitivo o que no pueden alejarse mucho de un lugar.

En primer lugar, al crear la clase, tenemos que especificar en nuestra clase que se implementen las interfaces *GoogleApiClient.onConnectionFailedListener, GoogleApiClient.ConnectionCallbacks, ResultCallback<Status>*. Estas interfaces reciben callbacks en caso de que la conexión con GooglePlay Services sea correcta, falle, o se suspenda. La última recibe el resultado exitoso o no al crear o borrar una geofence.

```
public class MonitorizacionPacienteActivity extends AppCompatActivity implements
        GoogleApiClient.ConnectionCallbacks, GoogleApiClient.OnConnectionFailedListener, ResultCallback<Status> {
```

Figura 41. Interfaces de MonitorizacionPacienteActivity

Para poder conectarnos a la API de Google es necesario crear una instancia de *GoogleApiClient*. En el siguiente método creamos un cliente Google. Este método lo llamamos al iniciar el Activity.

```
    protected synchronized void buildGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addConnectionCallbacks(this)
    .addOnConnectionFailedListener(this)
    .addApi(LocationServices.API)
    .build();
}
```

Figura 42. Creación de la instancia de GoogleApiClient

Para conectarse solo es necesario llamar al método *connect()* que se muestra en la Figura 43.

```
mGoogleApiClient.connect();
```

Figura 43. Conexión a la API de Google

En el momento en el que se tenga conexión con la API de Google y el usuario inicie la monitorización podremos crear el Geofence, añadiendo primero sus características. Para conocer la latitud y longitud, se usa la localización del paciente al crear el usuario, obtenida con la API de Google e insertada en la base de datos al crear el usuario. Se realiza una petición al servicio con la URI */mostrarLocGeofence* para recuperar la localización. El radio se fija en 200 metros.

```
mGeofenceList.add(new Geofence.Builder()
        // Id para identificar el geofence.
        .setRequestId("geofence")

                // Región circular.
        .setCircularRegion(
                latitud, longitud,
                radio
        )
                // Tiempo de expiración.
        .setExpirationDuration(Geofence.NEVER_EXPIRE)

                // Tipos de transición.
        .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
                Geofence.GEOFENCE_TRANSITION_EXIT)

        .build());
```

Figura 44. Creación del Geofence

Si tenemos conexión con la API de Google añadimos el Geofence:

```
if (!mGoogleApiClient.isConnected()) {
    Toast.makeText(this, "no conectado", Toast.LENGTH_SHORT).show();
    return;
}
try {
    LocationServices.GeofencingApi.addGeofences(
            mGoogleApiClient,
            // The GeofenceRequest object.
            getGeofencingRequest(),
            // PendingIntent que se ejecutara al detectar una transación
            getGeofencePendingIntent()
    ).setResultCallback(this); // Resultado procesado en onResult().
} catch (SecurityException securityException) {
    // Excepcion generada si la app no usa el permiso ACCESS_FINE_LOCATION.
    logSecurityException(securityException);
}
```

Figura 45. Adición del Geofence

Cuando el usuario pulsa en el botón de Finalizar servicio se usa el método elimina el Geofence (Figura 46).

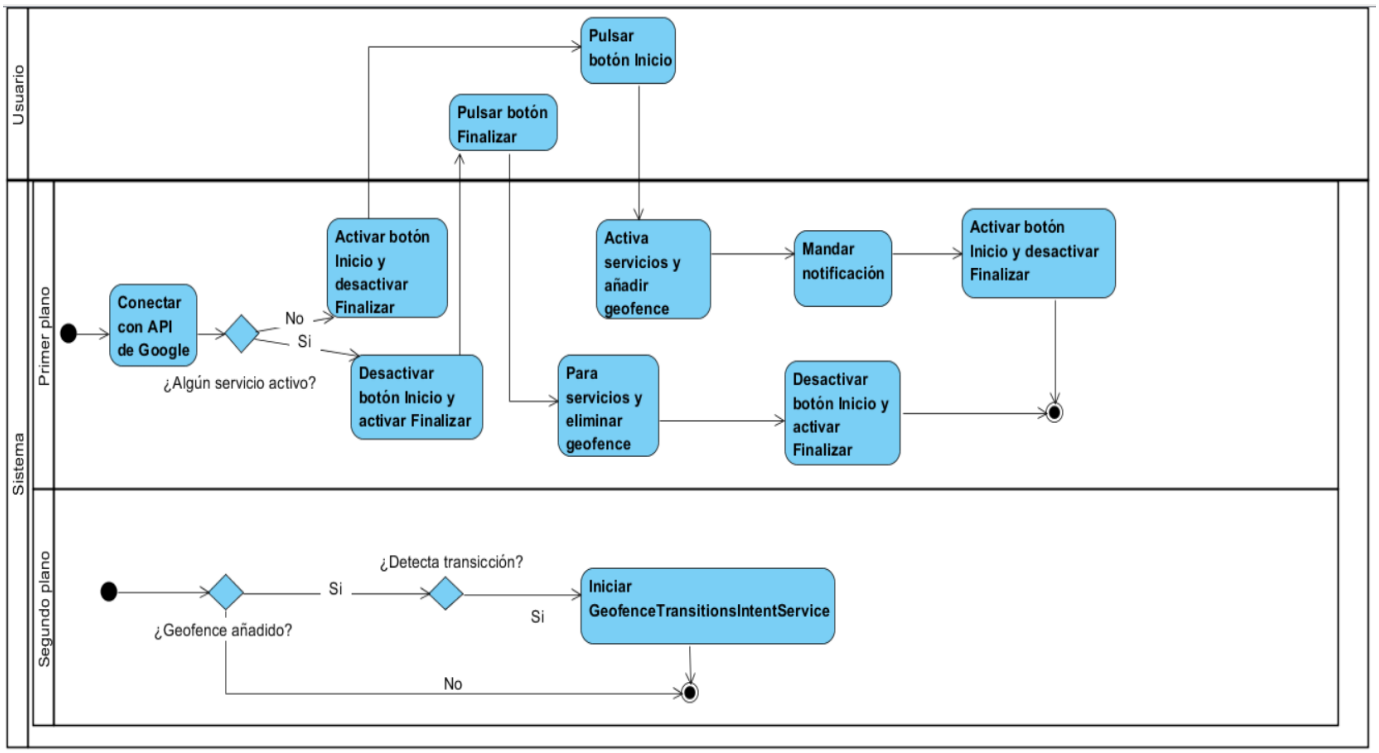


Figura 46. Diagrama de actividad de MonitorizacionPacienteActivity

3.4.6.2 GeofenceTransitionsIntentService

Esta clase es un *IntentService* que trata la detección de una transición de geofencing. Hay dos tipos de transiciones: Entrar y Salir (Figura 47).

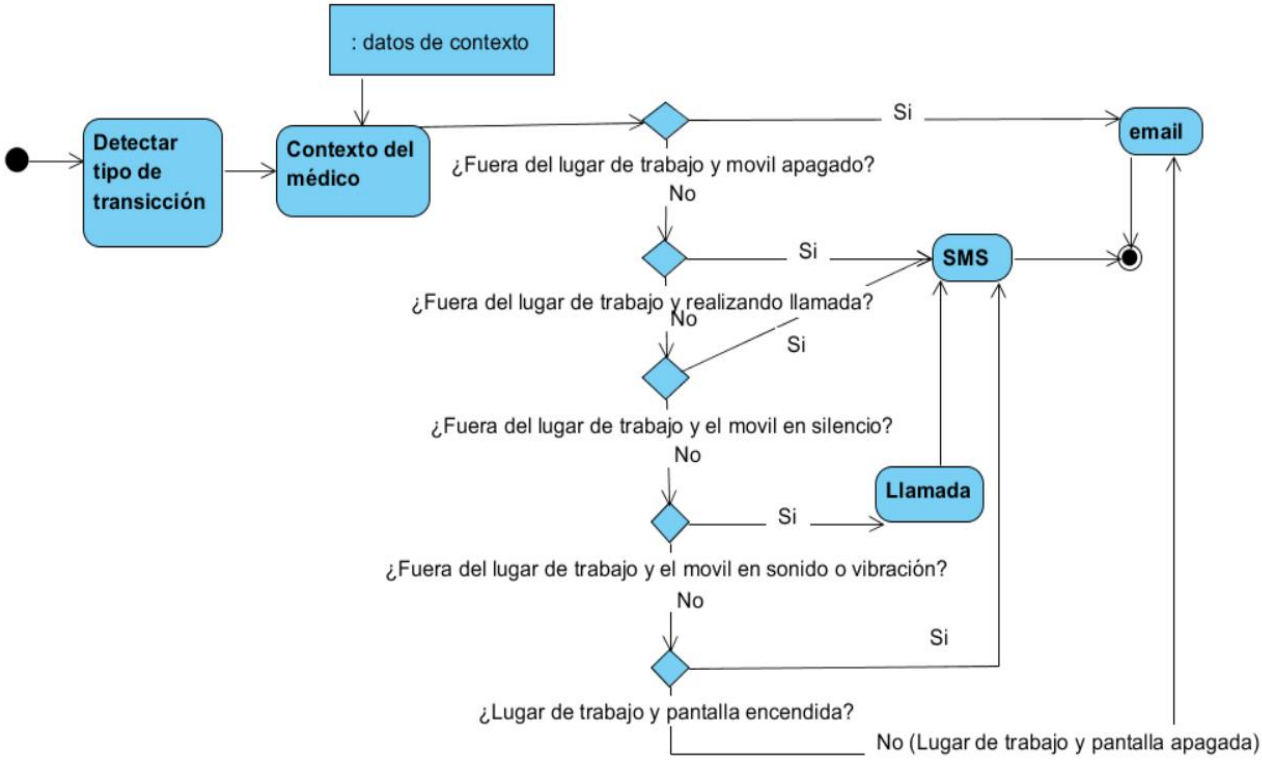


Figura 47. Diagrama de actividad de GeofenceTransitionsIntentService

Se envía un aviso al médico designado, necesitando conocer el contexto del médico. Se utilizan cinco peticiones al servicio para conocer su modo de audio, estado de la pantalla, proximidad, llamada y su localización con los recursos (*mostrarLastLocalizacion.php, mostrarAudio.php, mostrarPantalla.php, mostrarLlamda y mostrarProximidad.php)* así como el recurso *mostrarEmailTelefono.php* para conocer su email y teléfono. Con estos datos y según los tipos de notificaciones explicados en el apartado 3.4 se procede a conocer la distancia entre la localización actual del médico y su lugar de trabajo prefijado, el cual se consulta con la URI /*mostrarLocTrabajo*. Según el resultado la distancia se considera si se encuentra cerca o lejos.

Según la fecha de los datos de contexto del médico, se sabe si ha estado mucho tiempo sin monitorizar. Si esta fecha es mayor que una hora ignoramos los demás datos y se considera el móvil como apagado.

Las tres opciones de notificación son:

1. Enviar un email: siendo email el email obtenido de la base de datos correspondiente al médico, y *emailorigen* y contraseña los configurados para que sean siempre el origen de los emails (Figura 48).

```
Mail m = new Mail(emailorigen, contasena);
String[] toArr = {email};
m.setTo(toArr);
m.setFrom(emailorigen);
m.setSubject("Geofence");
m.setBody(notificationDetails);
try {
    m.send();

} catch(Exception e) {
    Log.e(TAG, "Could not send email", e);
}
```

Figura 48. Envío de email

Se necesita crear una clase *Mail* extensión de *javax.mail.Authenticator* que configure la autentificación, servidor de salida y el envío del mensaje. Para esto se necesita incluir tres librerías de Java.

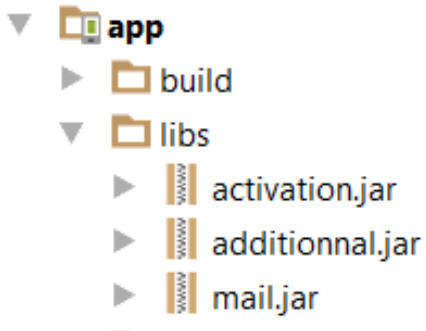


Figura 49. Librerías de Java para envío de email

2. Enviar un SMS: siendo el teléfono el obtenido en la base de datos correspondiente al médico (Figura 50).

```
String phoneNo = telefono;
String sms = notificationDetails;

try {
    SmsManager smsManager = SmsManager.getDefault();
    smsManager.sendTextMessage(phoneNo, null, sms, null, null);

} catch (Exception e) {
    e.printStackTrace();
}
```

Figura 50. Envío de SMS

3. Llamar: siendo el teléfono el obtenido en la base de datos correspondiente al médico (Figura 51).

```
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:" + telefono));
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
try {
    startActivity(intent);
} catch (Exception e) {

    e.printStackTrace();
}
```

Figura 51. Llamada

3.4.7 MapsActivity

Tras finalizar la creación del usuario como médico se inicia esta clase, que utiliza el GoogleMaps para que el usuario pueda identificar su lugar de trabajo fácilmente (Figura 52). Para esto se implementan las interfaces *OnMapReadyCallback* (para saber cuándo ha terminado de cargarse el mapa), *OnMapClickListener* (para detectar cuando el usuario pulsa en algún lugar del mapa) y las necesarias para conectarnos a la API de Google.

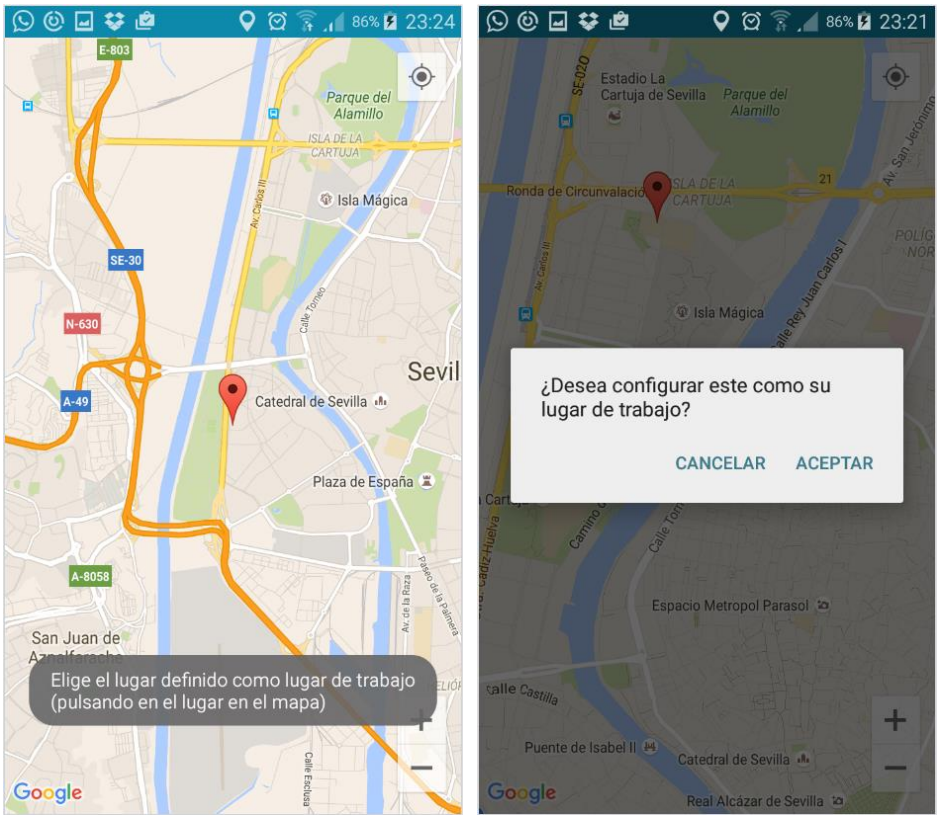


Figura 52. Layout de MapsActivity

```
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback,
        GoogleMap.OnMapClickListener,
        GoogleApiClient.ConnectionCallbacks,
        GoogleApiClient.OnConnectionFailedListener {
```

Figura 53. Interfaces de MapsActivity

Lo primero es conectarse a la API de Google, obtener el soporte del mapa y obtener notificación cuando el mapa esté listo para usarse (Figura 54).

```
SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
        .findFragmentById(R.id.map);
mapFragment.getMapAsync(this);
```

Figura 54. Creación del mapa

Cuando el mapa está listo y se está conectado a la API de Google se obtiene la localización actual del usuario y se crea una marca en el lugar (Figura 55).

```
marca = mMap.addMarker(new MarkerOptions()
        .position(position)
        .title("Lugar de trabajo")
        .draggable(true));
mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(position, 13));
```

Figura 55. Creación de marca

Cuando el usuario pulse en algún lugar del mapa, se le preguntará con una alerta si ese es el lugar que desea configurar como su lugar de trabajo. Si ese así, se actualiza el usuario en la base de datos mediante la petición al servicio usando el recurso *insertarLocTrabajo.php* (Figura 56).

Al finalizar se inicia la clase *ListaPacientesMedicoActivity* pasándole como parámetro el DNI del usuario.

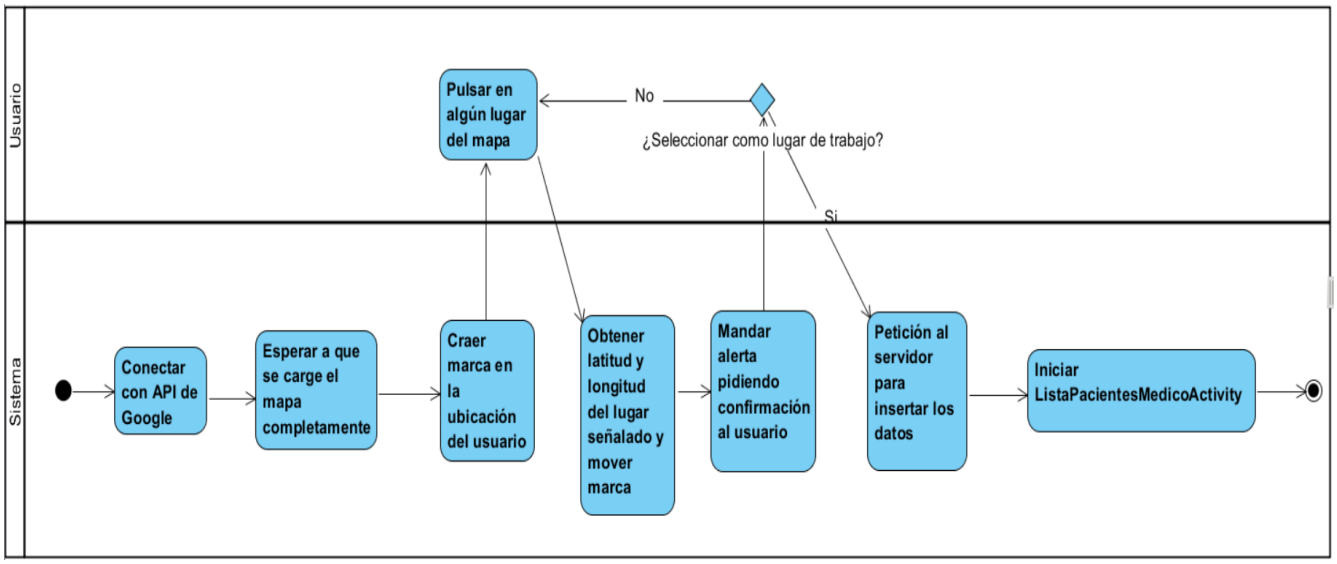


Figura 56. Diagrama de actividad de MapsActivity

3.4.8 ListaPacientesMedicoActivity

Esta clase hace uso de un ListView para mostrar el listado de los pacientes (Figura 57). Para esto se necesita un adaptador, la clase *Lista_adaptador* tiene los métodos necesarios para crear la lista y reaccionar al pulsar un elemento. También es necesaria la clase *Usuario* para personificar cada elemento de la lista.

Al iniciar esta clase se realiza una petición al servidor con el método *mostrarNombresPacientes.php* con cuyo resultado crearemos la lista (Figura 58). Este proceso puede tardar un tiempo mayor de lo esperado en el caso de haber muchos pacientes, pero para hacer más cómoda la espera al usuario en caso de que fuera necesario se ha implementado una *progressBar* (barra de progreso) circular.

Figura 57. Layout de ListaPacientesMedicoActivity

Cuando el usuario pulsa sobre uno de los elementos de la lista se llama al método *ElementoLista(View view)*, el cual inicia la siguiente actividad *InformacionPacienteActivity* pasándole como parámetros el DNI y el nombre del paciente seleccionado así como el DNI del médico que está actualmente activo.

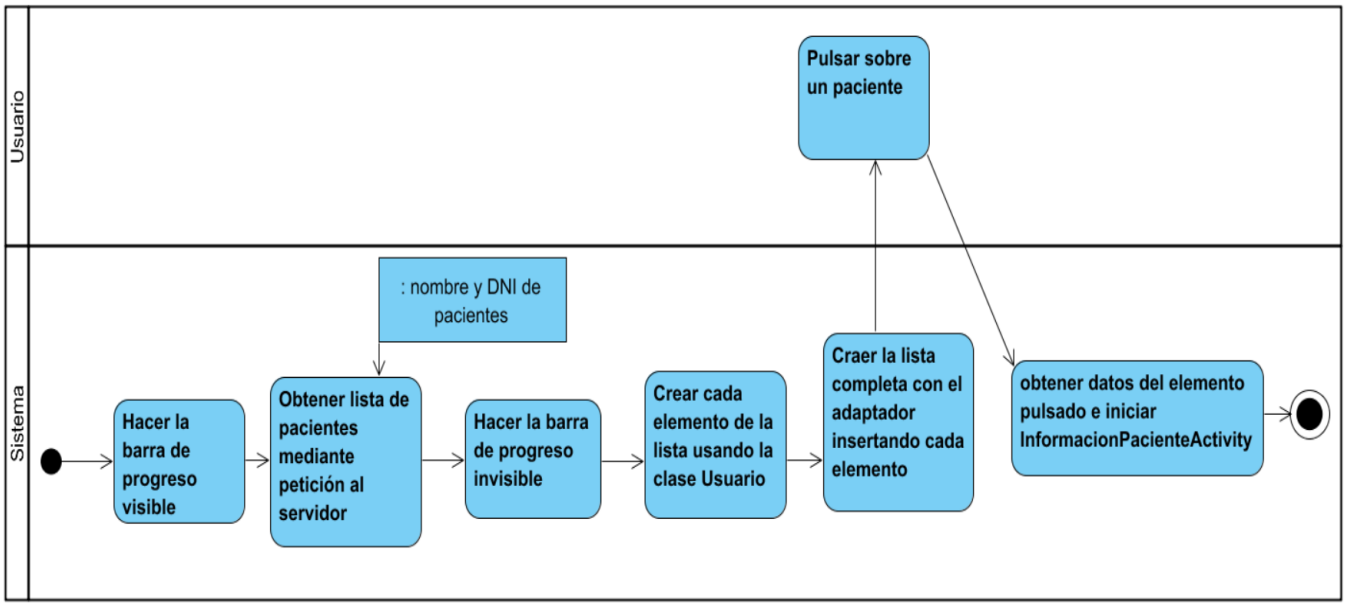


Figura 58. Diagrama de actividad de ListaPacientesMedicoActivity

3.4.9 InformacionPacienteActivity

En esta clase se encuentra otro formulario con el que el médico puede consultar datos de un paciente (Figura 59). El formulario se compone de una lista desplegable donde se podrá seleccionar el tipo de dato que quiere consultar, es decir la tabla correspondiente en la base de datos (Figura 62), un *DatePicker* (selector de fecha) que aparece al pulsar sobre el texto en un fragmento como un calendario para elegir la fecha desde cuando hacer la consulta y un *TimePicker* (selector de hora) igual que el de fecha para seleccionar la hora desde la que hacer la consulta (Figura 60). Si no se selecciona fecha ni hora se obtendrán todos los datos.

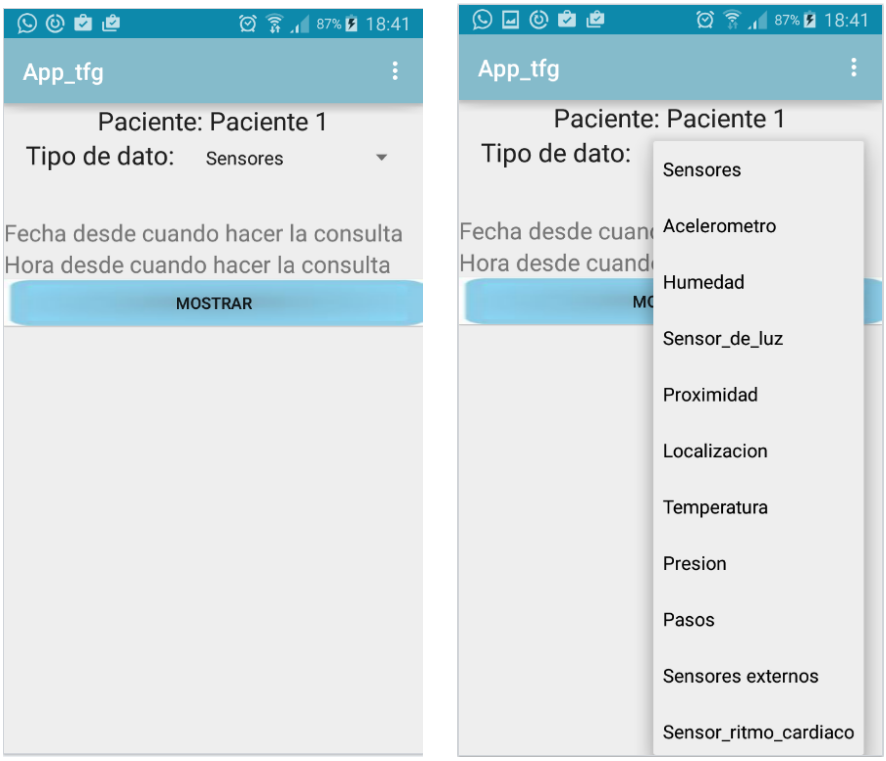


Figura 59. Layout de InformacionPacienteActivity

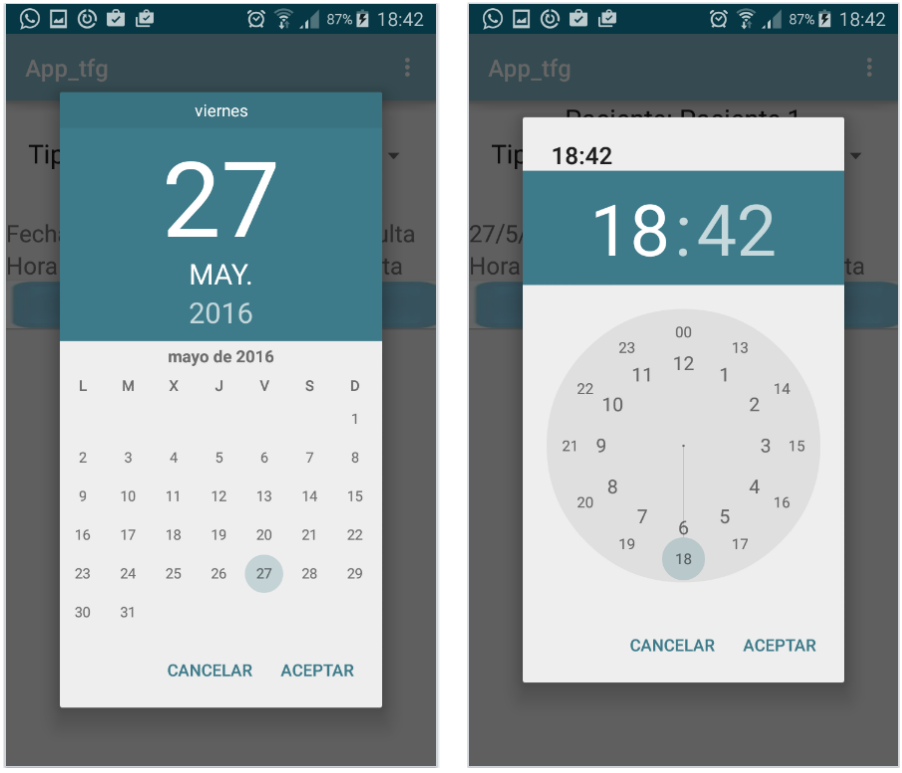


Figura 60. Selectores de fecha y hora

Una vez que el usuario pulsa sobre *Mostrar* se llama al método *mostrarDatos(View view)*. Antes de realizar la petición al servidor, se hace visible una *progressBar* al igual que en la clase anterior por si tarda más de lo esperado. En la petición al servidor se llama al método *mostrarDatosFecha.php*, devolverá tantas filas de datos

como encuentre según la solicitud, Tras volver a ocultar la barra de progreso se imprimen en un *scrollView* todas las filas recibidas (Figura 61).

Figura 61. Listado de dato

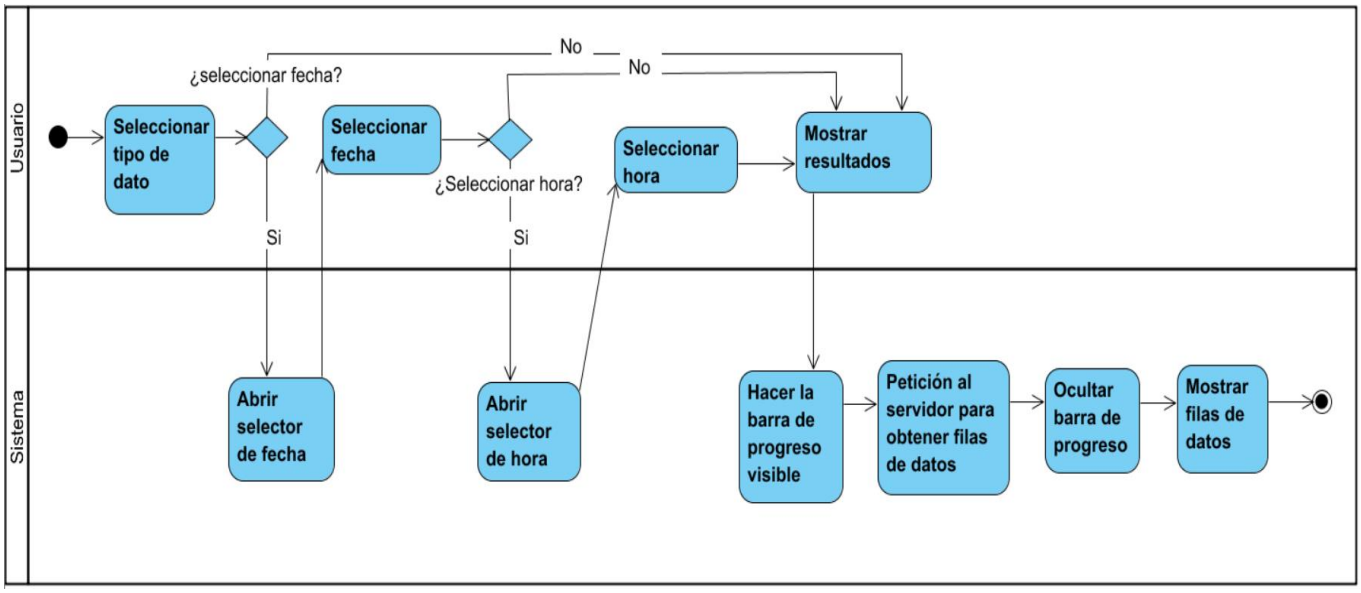


Figura 62. Diagrama de actividad de InformacionPacienteActivity

3.4.10 Menú del médico

Desde que el médico inicia sesión dispone de un menú en cada una de los *Activities*, este menú contiene 4 ítems (Figura 63): pacientes, configuración, monitorización y cerrar sesión.

- Pacientes: opción que permite volver a la pantalla de listado de pacientes.
- Configuración: opción que inicia la clase *ConfiguracionActivity*.
- Monitorización: opción que inicia la clase *MonitorizacionMedicoActivity*.
- Cerrar sesión: opción que permite volver al inicio, si los servicios de la monitorización están activos, es decir no se ha detenido la monitorización, se paran todos los servicios llamando al método *Stop_Servicios()* antes de cerrar sesión.

Figura 63. Menú del médico

3.4.10.1 ConfiguracionActivity

Esta clase permite al médico cambiar el email en el cual recibe las notificaciones (Figura 64). Al pulsarse cambiar, el método *enviarDatosConf (View view)* comprueba que el email contenga un '.' Y una @ (Figura 65). Si es así hace una petición al servicio con la URI */modificarMedico*. Al modificarse se avisa del éxito al usuario con un Toast y vuelve a la clase *ListaPacientesMedicoActivity*.

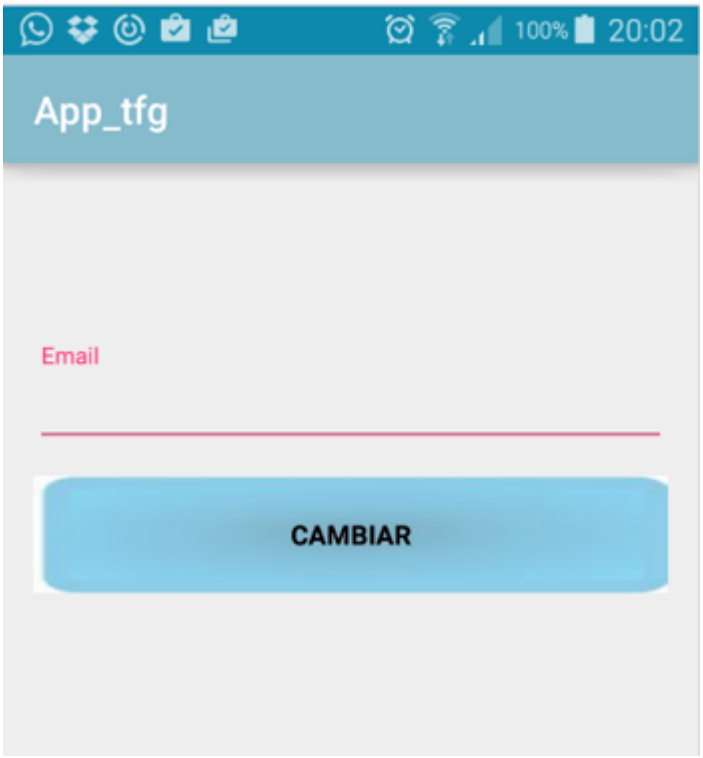


Figura 64. Layout de ConfiguracionActivity

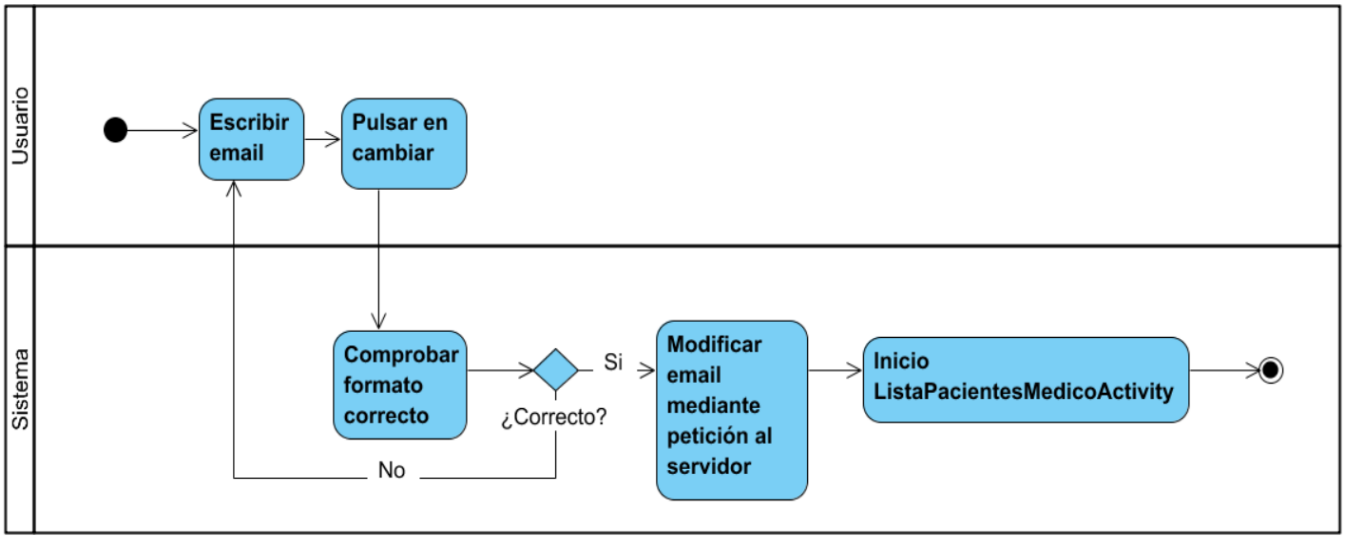


Figura 65. Diagrama de actividad de ConfiguracionActivity

3.4.10.2 ConfiguracionPacienteActivity

De forma similar a la clase anterior, esta ejecuta el método *enviarDatosConf(View view)* (Figura 66*)* cuando el usuario pulsa en *cambiar*, se comprueba que al menos uno de estos campos esté relleno y se actualizan los datos mediante la petición al servicio con la URI */modificarPaciente* (Figura 67). Si finaliza todo correctamente se vuelve a la clase *InformacionPacienteActivity*.

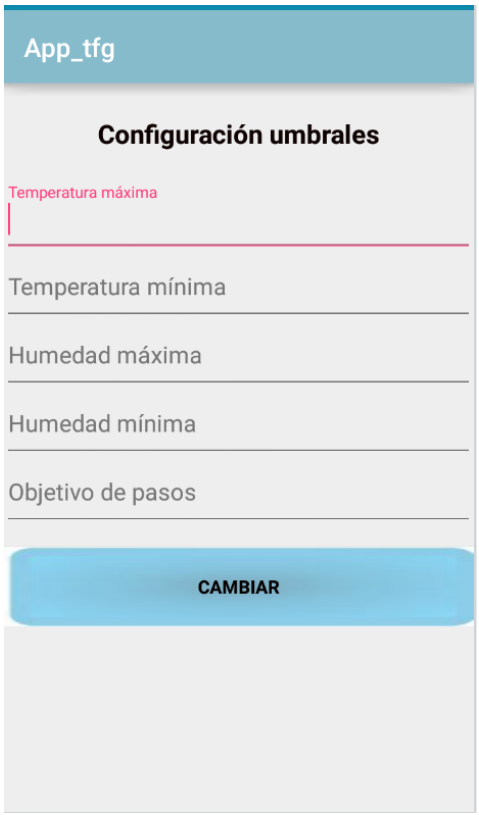


Figura 66. Layout de ConfiguracionPacienteActivity

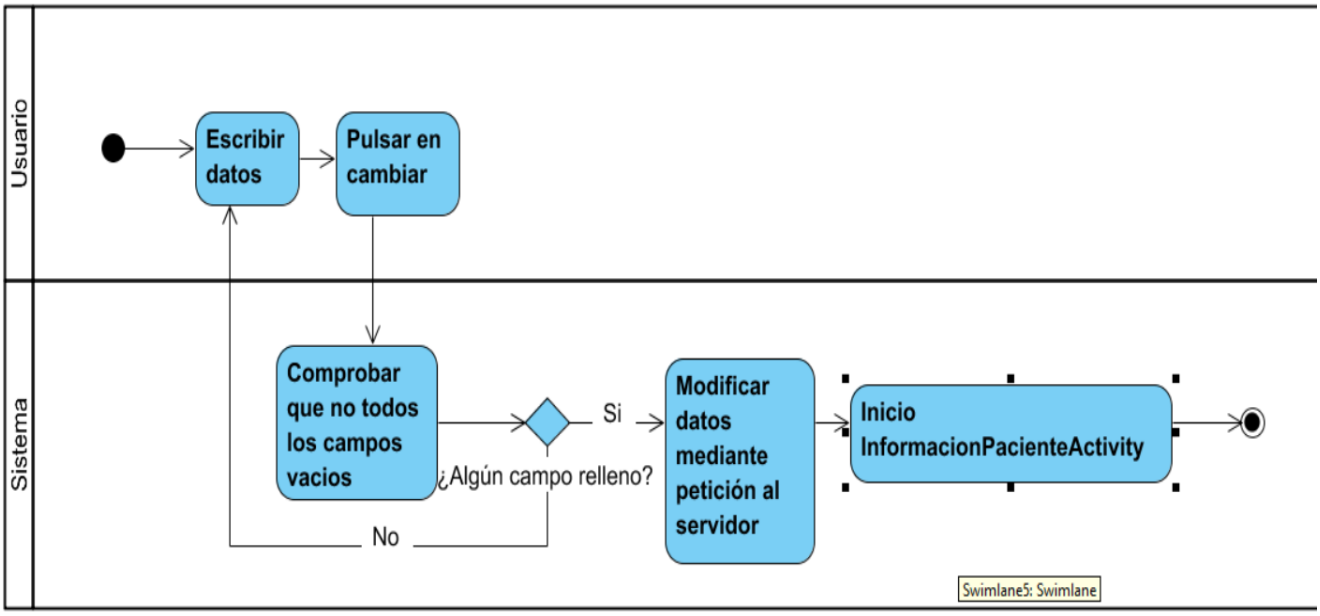


Figura 67. Diagrama de actividad de ConfiguracionPacienteActivity

3.4.10.3 MonitorizacionMedicoActivity

Funciona igual que *MonitorizacionPacienteActivity* solo que inicia menos servicios y no añade Geofencing (Figuras 68 y 69).

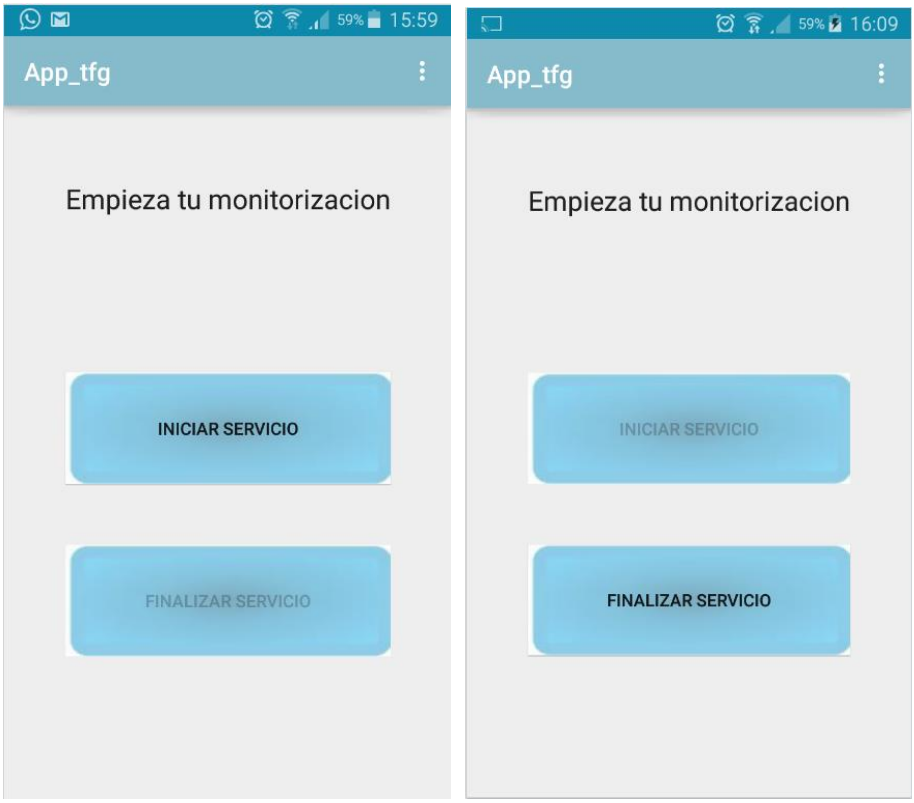


Figura 68. Layout de MonitorizacionMedicoActivity

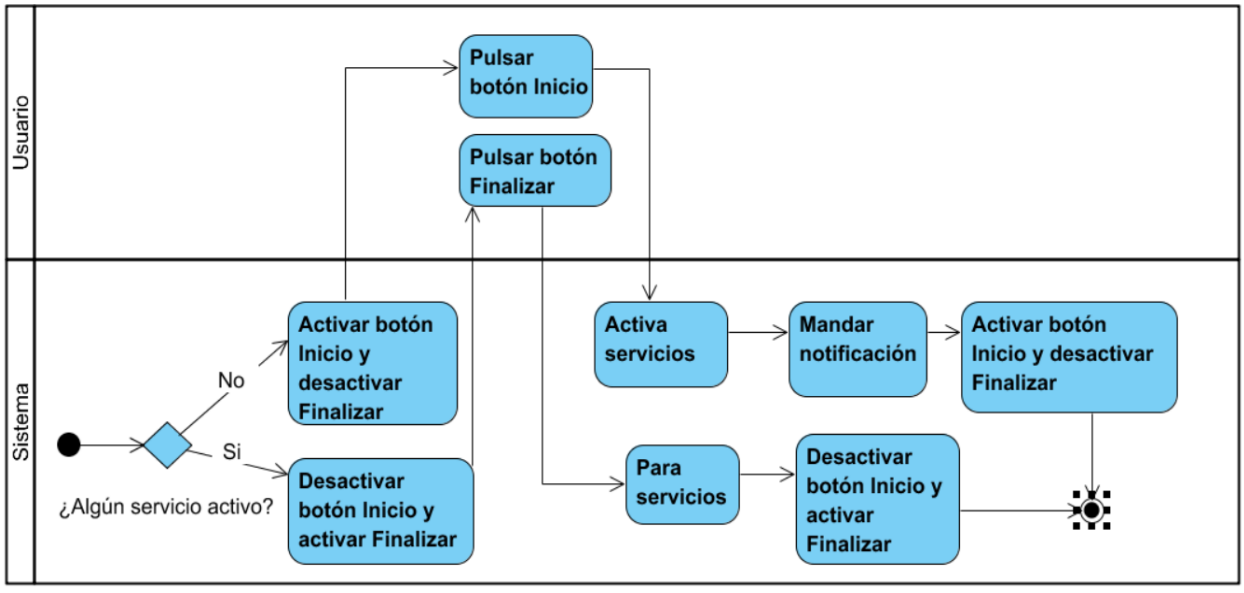


Figura 69. Diagrama de actividad de MonitorizacionMedicoActivity

3.4.11 Servicios

Aparte de *GeofenceTransitionsIntentService* se decidió usar la extensión de la clase Service en lugar de su subclase IntentService para la monitorización en segundo plano, ya que la principal diferencia es que el IntenService al acabar su tarea finaliza él solo, y lo que se desea es que solo finalice al indicárselo, como pasa con el Service.

Para *GeofenceTransitionsIntentService* si se usó el *IntentService* porque solo se llama al detectarse una transacción y su cometido es enviar la notificación.

Los servicios usados en este proyecto para la monitorización se describen a continuación.

3.4.11.1 AudioService

El audio no es un sensor, por lo que se monitorizará de manera distinta. Para ello se usa la clase *BroadcastReceiver* usando como filtro el cambio del modo de audio (Figura 70).

```
IntentFilter filter=new IntentFilter(
        AudioManager.RINGER_MODE_CHANGED_ACTION);
```

Figura 70. Filtro para cuando cambia el modo de audio

```
receiver=new BroadcastReceiver(){
    @Override
    public void onReceive(Context context, Intent intent) {
        audio();

    }
};
```

Figura 71. Creación de BroadcastReceiver

Para iniciarlo se utiliza el método de la Figura 72.

```
registerReceiver(receiver, filter);
```

Figura 72. Iniciar BroadcastReciver

Cada vez que se detecta un cambio en el modo de audio se llama al método *audio(),* que obtiene el nuevo modo de audio y realiza una petición al servicio web usando el recurso *insertarAudio.php* para insertar el dato. Al detener la monitorización y parar el servicio se llama al método *unregisterReceiver* (Figura 73).

```
@Override
public void onDestroy()  {
    unregisterReceiver(receiver);
}
```

Figura 73. Detener BroadcastReciver

3.4.11.2 ScreenService

Este servicio es prácticamente igual que el anterior, lo único que cambia es el tipo de filtro de *BroadcastReceiver*, necesita dos tipos: ACTION_SCREEN_ON y ACTION_SCREEN_OFF.

En la petición al servicio web para insertar los datos se usa el método *insertarPantalla.php.*

3.4.11.3 CallService

Este servicio también es del estilo de los anteriores, usando el tipo de filtro de *BroadcastReceiver:* ACTION_PHONE_STATE_CHANGED.

En la petición al servicio para insertar los datos se usa el recurso *insertarLlamada.php.*

3.4.11.4 LocationService

Este servicio tampoco usa sensores. Para capturar la localización se vuelve a usar la API de Google junto con algunas clases de Android.Location.

Se especifica en nuestra clase que se implemente las interfaces *GoogleApiClient.onConnectionFailedListener, GoogleApiClient.ConnectionCallbacks* y *LocationListener.*

```
public class LocationService extends Service implements
        GoogleApiClient.ConnectionCallbacks,
        GoogleApiClient.OnConnectionFailedListener, LocationListener {
```

Figura 74. Interfaces de LocationService

Las dos primeras para la conexión a la API de Google y la última para escuchar el cambio de localización. Lo primero es crear la instancia de *GoogleApiClient* y conectarse a ella como se explicó en el apartado 3.4.6.1. Antes de conectarnos se comprueba que esté el GPS conectado.

Tras esto se crea una instancia *LocationRequest* para configurar la prioridad con la que se pedirá la localización así como el máximo y mínimo intervalo de tiempo para ello (Figura 75).

```
mLocationRequest = LocationRequest.create()
        .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
        .setInterval(120 * 1000)        // 2 minutos, in milliseconds
        .setFastestInterval(60 * 1000); // 1 minuto, in milliseconds
```

Figura 75. Creación de la instancia LocationRequest

En el momento en que se establezca la conexión con la API de Google se obtiene una primera localización y se llama al método *Posicion(location)* para tratarla. Luego se crea una actualización de la localización con la instancia *LocationRequest* anteriormente creada.

Cada vez que se detecte un cambio de localización se volverá a llamar al método *Posicion.* Al parar el servicio se desconecta de la API.

El método *Posicion* utiliza las clases Geocoder y Address de Android.Location para conocer más datos sobre la localización con la latitud y longitud (Figura 76).

```
geocoder = new Geocoder(this, Locale.getDefault());

addresses = geocoder.getFromLocation(loc.getLatitude(), loc.getLongitude(), 1);

final String address = addresses.get(0).getAddressLine(0);
final String city = addresses.get(0).getLocality();
final String state = addresses.get(0).getAdminArea();
final String country = addresses.get(0).getCountryName();
final String postalCode = addresses.get(0).getPostalCode();
```

Figura 76. Obtención de más datos de una localización

Tras esto se realiza la petición al servicio web para subir los datos con la URI */insertarLocation.*

3.4.11.5 LigthService

Los servicios que quedan sí están basados en sensores, por lo que todos tendrán la misma manera de capturar datos. Es necesario implementar la interfaz *SensorEventListener* para la escucha de eventos de sensores (Figura 77).

```
public class LightService extends Service implements SensorEventListener {
```

Figura 77. Interfaz para la escucha de sensores

Al empezar el servicio se determina el tipo de sensor a capturar, en este caso el sensor de luz, definido por TYPE_LIGHT. También se define el registro de escucha con el tipo de sensor y la frecuencia de escucha (Figura 78).

```
mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
```

Figura 78. Registrar la escucha del sensor

En el método *OnSensorChanged (SensorEvent event),* llamado cada vez que se detecte un cambio en el sensor, se captura el valor del sensor.

Este sensor cambia constantemente ya que detecta cada pequeño cambio en la intensidad de luz del ambiente, por eso no se puede insertar todos los eventos detectados y se realiza la inserción cada minuto. Para controlar si ha transcurrido un minuto, se comprueba el tiempo pasado entre el último dato insertado y el evento capturado. Si es mayor que un minuto se realiza la petición al servicio web para insertar datos con la URI */insertarLuz* y se actualiza la fecha de la última inserción. Se inicializan las variables al comienzo del servicio para que la primera vez que se capte un evento se inserten datos. Al detener el servicio se dejan de registrar los eventos del sensor (Figura 79).

```
@Override
public void onDestroy()  {

    mSensorManager.unregisterListener(this);
}
```

Figura 79. Eliminación de la escucha de eventos del sensor

3.4.11.6 ProximityService

El sensor de proximidad se monitorizará igual que el de luz definiendo el tipo TYPE_PROXIMITY y usando el método *insertarProximidad.php.* Pero este sensor no cambia con tanta frecuencia por lo que no es necesario insertar datos controlando el tiempo entre inserciones, sino cada vez que este cambie.

3.4.11.7 PressureService

Este sensor también es como los anteriores usando el tipo de sensor TYPE_PRESSURE. Realizando la petición al servicio con la URI */insertarPresion.* El barómetro sí cambia con frecuencia por lo que se usa el mismo método que la luz.

3.4.11.8 HumidityService

Este sensor es el tipo TYPE_RELATIVE_HUMIDITY. A cada minuto se usa el recurso *insertarHumedad.php* del servicio web. Para obtener los umbrales de humedad en los que saltaría la notificación se usa el recurso del servicio *mostrarHumedad.php*. Si alguno de los dos o los dos no son cero, se usan para mandar una notificación si la humedad es mayor que la humedad máxima o menor que la mínima durante dos actualizaciones seguidas de datos y no se han realizado notificaciones en la última hora. Para la notificación del médico se realiza el mismo procedimiento que en el apartado 3.4.6.2.

3.4.11.9 TemperatureService

Sensor detectado con el tipo TYPE_AMBIENT_TEMPERATURE. Inserta el dato cada minuto con la URI */insertarTemperatura* del servicio. Realiza la detección de incidencias y el envío de la notificación al igual que con el sensor de humedad, pero obteniendo los umbrales mediante el recurso *mostrarTemperatura.php*.

3.4.11.10 StepCounterService

A partir de la versión de Kitkat, Android soporta el sensor de contador de pasos, aunque no todos los dispositivos disponen de este sensor. Existe la alternativa de usar el acelerómetro de forma que usando los movimientos del dispositivo se detecta un movimiento periódico y lo registra como pasos.

La decisión fue implementar las dos opciones, ya que siendo la primera más precisa también se considera la alternativa de no disponer del sensor (Figura 80).

Primero se comprueba si existe el sensor detector de pasos, si así es se procede al igual que con los demás sensores, sumando un paso cada vez que salta el evento e insertando los datos a cada minuto usando el recurso *insertarPasos.php*. Si no se dispone del sensor de pasos, se registra la medición del acelerómetro.

```
mStep = mSensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR);
if(mStep==null){
    mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_FASTEST);
}else{
    mSensorManager.registerListener(this, mStep, SensorManager.SENSOR_DELAY_NORMAL);
}
```

Figura 80. Sensor detector de pasos y acelerómetro

Para el algoritmo de detección de pasos, aunque se encontraron bastantes artículos acerca de ello [26] [27], el desarrollo se basó en el proyecto de código abierto creado por Google llamado Podometer [28]. Este proyecto básicamente usa los datos del acelerómetro, encuentra máximos y mínimos y si la diferencia es mayor que un valor (el cual depende de la sensibilidad) entonces cuenta un paso, esto junto con algo de optimización sacado de pruebas de ejecución.

Se añadió la comprobación de falsos positivos, es decir que hasta que no se detecten varios pasos seguidos no se empiezan a contar como pasos.

Se consulta si el médico ha programado un máximo de pasos cada 24 horas con el recurso *mostrarPasos.php*. En caso de que así sea se programa la alerta de una notificación del mismo modo que en *GeofenceTransitionsIntentService* en el apartado 3.4.6.2 si se llega a este umbral.

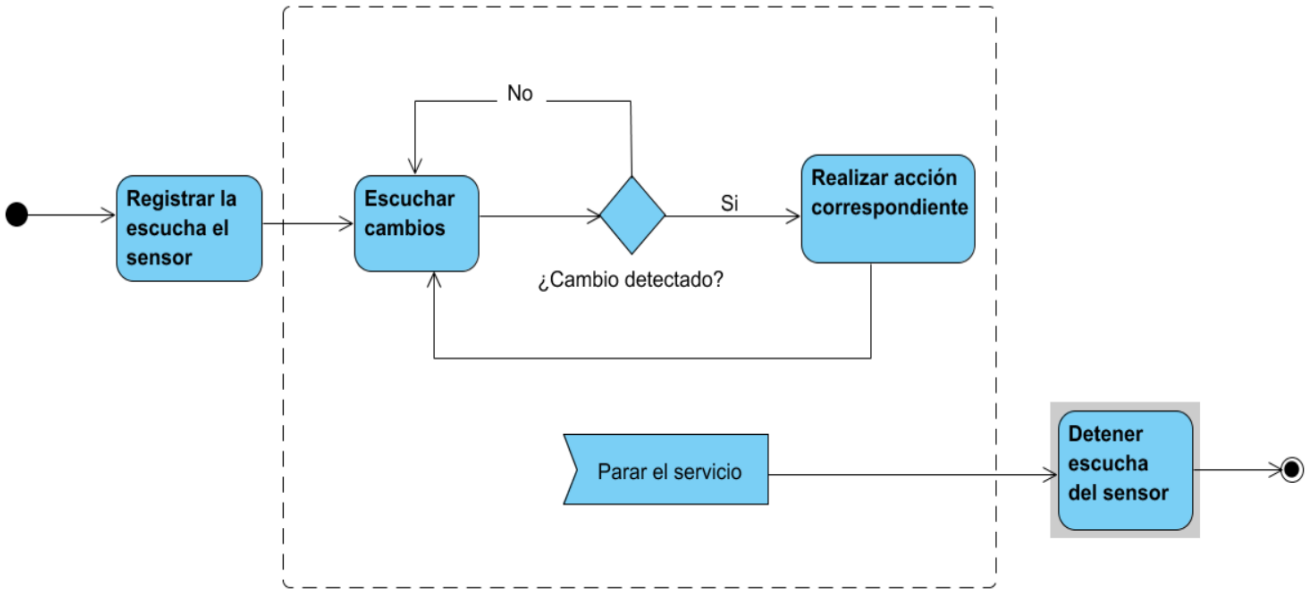


Figura 81. Diagrama de actividad de los servicios

3.4.12 Permisos necesarios

Para proteger ciertos recursos y características especiales, Android define un esquema de permisos. Toda aplicación que acceda a estos recursos está obligada a declarar su intención de usarlos. En caso de que una aplicación intente acceder a un recurso del que no ha solicitado permiso, se generará una excepción de permiso y la aplicación será interrumpida inmediatamente.

Cuando el usuario instala una aplicación este podrá examinar la lista de permisos que solicita la aplicación y

decidir si considera oportuno instalar dicha aplicación.

En esta aplicación se ha necesitado declarar los siguientes permisos:

- android.permission.INTERNET: Permite tener acceso a internet.
- android.permission.ACCESS_FINE_LOCATION: Permite localización GPS, basada en telefonía movil y Wi-Fi detallada.
- com.google.android.providers.gsf.permission.READ_GSERVICES: Permite el acceso hacia los servicios web de Google.
- android.permission.ACCESS_LOCATION_EXTRA_COMMANDS: Permite a una aplicación acceder a comandos adicionales de los proveedores de localización.
- android.permission.READ_LOGS: Permite el acceso a los logs creados por el sistema. Solo usado para el desarrollo de la app.
- android.permission.ACCESS_NETWORK_STATE: Permite obtener información sobre todas las redes.
- android.permission.WRITE_EXTERNAL_STORAGE: Permite escribir en almacenamiento externo.
- android.permission.ACCESS_BACKGROUND_SERVICE: Permite ejecutar un servicio en segundo plano.
- android.permission.SEND_SMS: Permite la aplicación mandar de texto SMS sin la validación del usuario.
- android.permission.CALL_PHONE: Permite realizar llamadas sin la intervención del usuario.
- android.permission.READ_PHONE_STATE: Permite consultar identidad y estado del teléfono.
- android.permission.BODY_SENSORS: Permite el acceso a los datos de los sensores que están monitorizando el cuerpo del usuario.

Estos permisos deben declararse en el archivo *AndroidManifest.xml.*

Por último, mostramos un dato que es necesario introducir en el fichero manifiesto para poder ver los mapas de Google, y es el valor de una clave de Google que se obtiene a través de su consola de desarrollador (Figura 82). Esta clave una vez obtenida puede utilizarse en distintas aplicaciones que necesiten incorporar mapas.

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="@string/API_KEY" />
```

Figura 82. Clave de Google para poder usar mapas

3.5 Ejemplos de uso

En este apartado se verán distintas pruebas realizadas para el correcto funcionamiento de la aplicación. Especificar que los dispositivos usados para las pruebas no disponen de sensores de humedad y temperatura, pero su funcionamiento sería del estilo de los demás sensores.

- Ejemplo para crear un usuario médico: Al introducir todos los campos del médico y configurar el lugar de trabajo como se puede ver en la Figura 83, se introducen estos en la base de datos (Figuras 84 y 85):

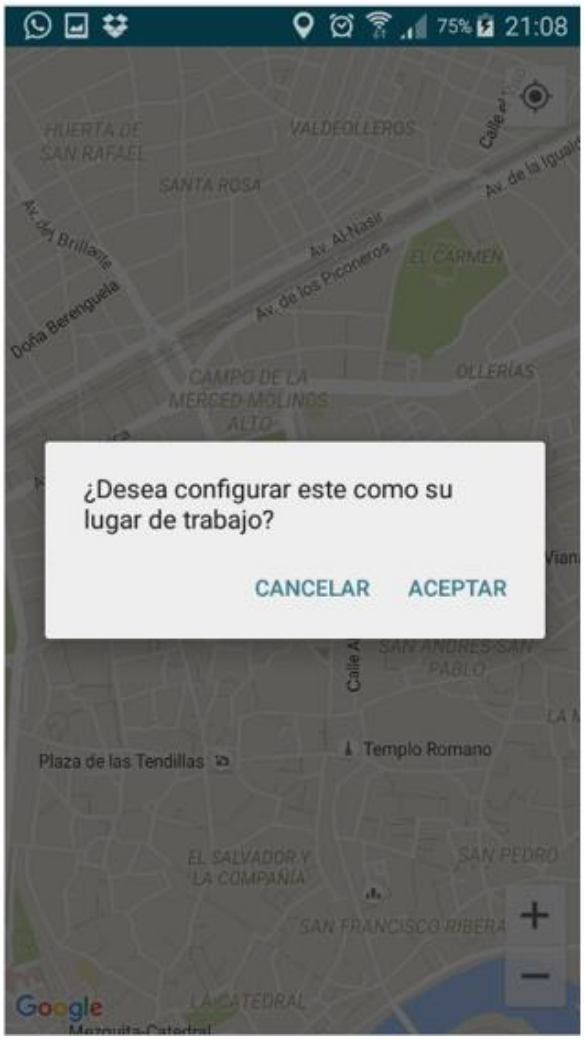


Figura 83. Crear nuevo médico

dni	contrasena
11111111a	1234

Figura 84. Usuario creado en la tabla usuario

usuario_dni	nombre	fecha_nacimiento	especialidad	email	telefono	latitud_trabajo	longitud_trabajo
11111111a	medico1	1888-01-01	cuidador	emmacamachoraya@gmail.com	+34690026877	37.88872908374616	-4.777045771479607

Figura 85. Médico creado en la tabla medico

Al crear el médico se empieza su monitorización y obtenemos como primeros datos los mostrados en la Figura 86.

Fecha 1	Modo_audio	usuario_dni
2016-06-18 17:58:39	Modo normal	11111111a

Fecha 1	Latitud	Longitud	Altitud	Precisionn	Pais	Estado	Ciudad	Codigo_postal	Direccion	usuario_dni
2016-06-18 17:58:45	37.8888828	-4.7775865	0	30	España	null	Córdoba	14001	Plaza de Colón, 4	11111111a

Fecha 1	modo_pantalla	usuario_dni
2016-06-18 17:58:39	On	11111111a

Fecha 1	Aproximacion	usuario_dni
2016-06-18 17:58:39	8	11111111a

Fecha 1	modo_llamada	usuario_dni
2016-06-18 17:58:39	Sin actividad	11111111a

Figura 86. Filas insertadas al monitorizar al médico

- Ejemplo para crear un usuario paciente asignado al médico anterior: Al introducir todos los campos del paciente como se puede ver en la Figura 87, se introducen estos en la base de datos (Figuras 88 y 89):

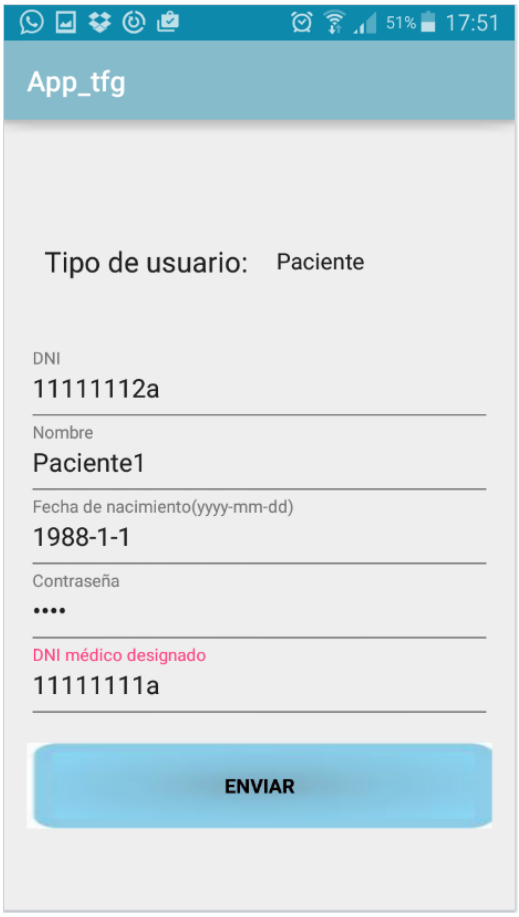


Figura 87. Crear nuevo paciente

dni	contrasena
11111112a	1234

Figura 88. Usuario creado en la tabla usuario

nombre	fecha_nacimiento	usuario_dni ▲ 1	medico_dni	latitud_geofence	longitud_geofence	temperatura_max	temperatura_min	humedad_max	humedad_min	pasos
Paciente1	1988-01-01	11111112a	11111111a	37.8888558	-4.7776598	0	0	0	0	0

Figura 89. Paciente creado en la tabla paciente

Al comenzar la monitorización del paciente se obtienen como primeros datos los mostrados en la Figura 90

Fecha ▼ 1	Latitud	Longitud	Altitud	Precisionn	Pais	Estado	Ciudad	Codigo_postal	Direccion	usuario_dni
2016-06-18 20:13:43	37.3831348	-6.0131078	0	30.623	España	null	Sevilla	41010	Calle Virgen de las Torres, 8	11111112a

Fecha ▼ 1	Modo_audio	usuario_dni
2016-06-18 20:13:23	Modo normal	11111112a

Fecha ▼ 1	modo_pantalla	usuario_dni
2016-06-18 20:13:23	On	11111112a

Fecha ▼ 1	Pasos	usuario_dni
2016-06-18 20:24:22	0	11111112a

Fecha ▼ 1	Presion	usuario_dni
2016-06-18 17:51:23	998.227	11111112a

Fecha ▼ 1	Aproximacion	usuario_dni
2016-06-18 20:13:34	8	11111112a

Fecha ▼ 1	Light	usuario_dni
2016-06-18 20:13:23	282	11111112a

Figura 90. Filas insertadas al monitorizar al paciente

- Ejemplo en el que un médico accede a los datos de un paciente desde la aplicación (Figuras 91 y 92). En la primera figura se observan los datos del audio, proximidad y del sensor de luz, en la segunda el modo de pantalla, localización y número de pasos insertados antes en la base de datos al monitorizar al paciente.

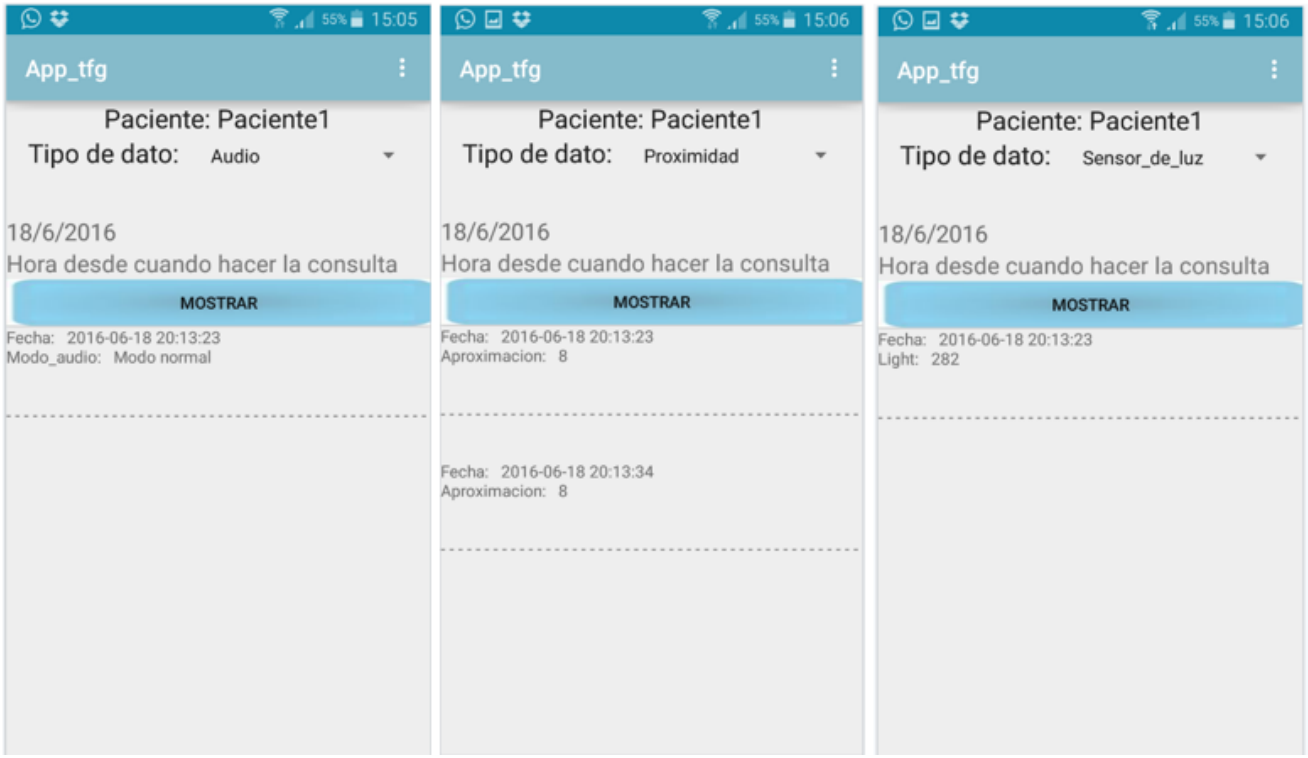


Figura 91. Datos de un paciente mostrados en la app

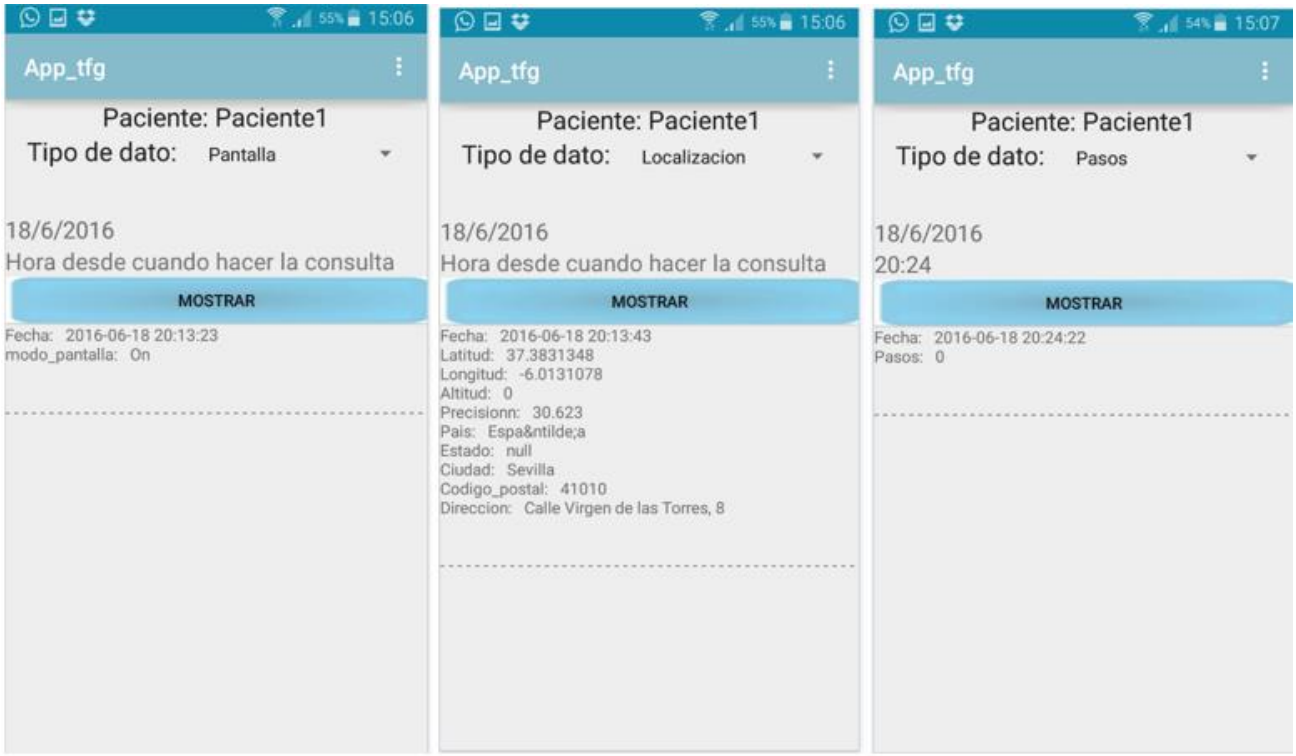


Figura 92. Datos de un paciente mostrados en la app

- Ejemplo en el que el médico configura un umbral de pasos para el paciente (Figuras 93 y 94). Se configura un objetivo de 20 pasos, cuando el paciente los sobrepase, se notificará al médico. También puede configurar los umbrales de temperatura y humedad.

Figura 93. Configuración del objetivo de pasos

nombre	fecha_nacimiento	usuario_dni	medico_dni	latitud_geofence	longitud_geofence	temperatura_max	temperatura_min	humedad_max	humedad_min	pasos
Paciente1	1988-01-01	11111112a	11111111a	37.8888558	-4.7776598	0	0	0	0	20

Figura 94. Inserción del objetivo de pasos en la fila del paciente

- Ejemplo en el que un paciente que necesita que se controle su localización y conocer si sale o entra de un perímetro prefijado, este entra en el perímetro o se encuentra dentro al iniciar la monitorización (por defecto también se enviará la notificación con el estado inicial del paciente con respecto al perimetro). Al mandar la notificación al médico, sus datos de contexto son uno de los siguientes: fuera del hospital y realizando llamada, fuera del hospital y el móvil en silencio o en el lugar de trabajo y pantalla encendida. Se le envía un SMS (Figura 95).

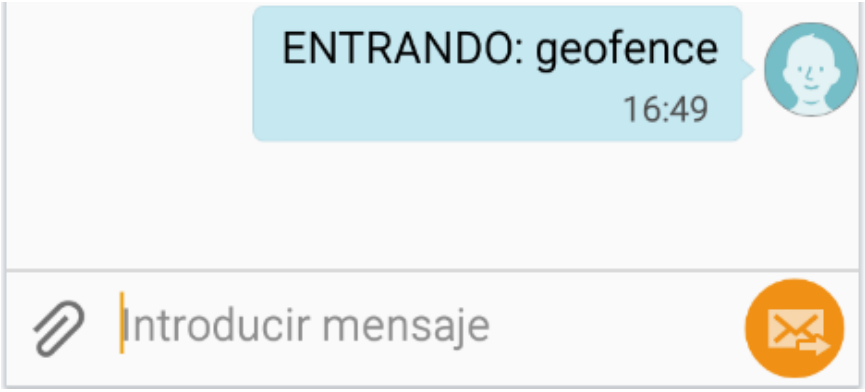


Figura 95. SMS enviado para notificar al médico

- Ejemplo en el que un paciente sale del perímetro prefijado. Al mandar al médico la notificación, sus datos de contexto son: fuera del hospital y móvil apagado (hace más de una hora que no se recibe

información de contexto) o en el lugar de trabajo y pantalla apagada o móvil apagado. Se le envía un email (Figura 96).

Figura 96. Email enviado para notificar al médico

- Ejemplo en el que un paciente supera el número de pasos necesarios desde que comenzó la monitorización (20 en el caso). Al mandar al médico la notificación, sus datos de contexto son: fuera del hospital y el móvil en sonido o vibración. Se realiza una llamada al médico y se le envía un SMS (Figura 97). Lo ideal habría sido que sonara una locución con la información en la llamada, pero al no encontrarse la manera de llevarlo a cabo, se decidió dejar la llamada como toque de atención y enviar el SMS con los datos.

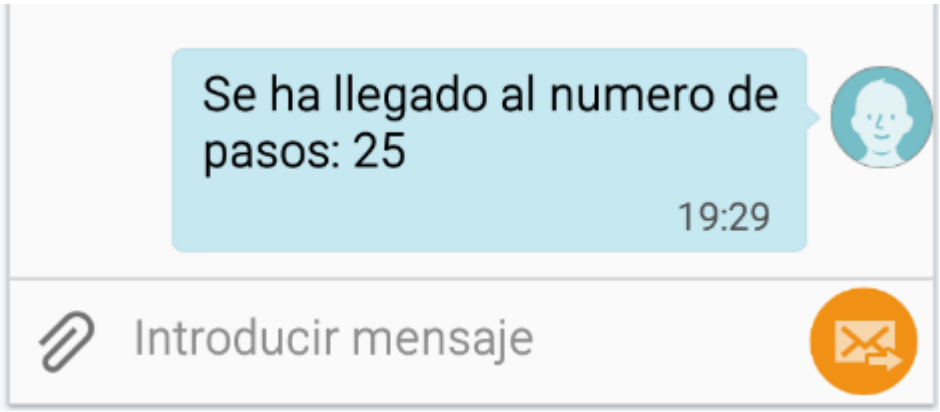


Figura 97. Envío del SMS tras la llamada

4 CONCLUSIONES Y LÍNEAS FUTURAS DE INVESTIGACIÓN

4.1 Conclusiones

Se comenzará la conclusión de este Trabajo de Fin de Grado con la mención de todas las cosas aprendidas. Por una parte, está la ampliación de conocimientos de lo adquirido en la carrera principalmente lo referido a Servicios Web, bases de datos, Android e ingeniería de software. Por otra parte, al ser un proyecto que relaciona la medicina y las tecnologías, se ha investigado acerca de este ámbito y aprendido lo importante que es el que a través de la ingeniería se ayude a la sanidad para obtener facilidades.

Hay un campo enorme que explotar con respecto a la monitorización a través de las nuevas tecnologías. Tradicionalmente, la atención sanitaria se basaba en la recogida de información del paciente exclusivamente cuando este se encontraba en presencia física del médico, pero hoy en día es posible obtener datos del día a día gracias al avance de las tecnologías y utilizarlos para reducir costes económicos, disminuir el tiempo entre visitas y tener un mayor control del paciente.

En este trabajo se desarrolla un Servicio web general para almacenar datos de los sensores de los dispositivos móviles, adaptado a la aplicación de ejemplo, pudiendo surgir distintas variantes para casos sanitarios más específicos. Los sensores pueden ser calibrados y usados según las necesidades. En esta aplicación de ejemplo se explotan sobre todo los de información medioambiental, localización y acelerometría. Este último se usa como detector de pasos, pero como conclusión personal, se podría usar para una variedad de situaciones, por ejemplo, detectar caídas. También es interesante no solo centrar la monitorización en el paciente, también en el profesional a cargo para, como se plasma en la aplicación, poder usar su contexto para el envío de las alertas. Con todo esto tenemos un proyecto que recoge y almacena información de algo muy innovador y en continuo progreso como son los sensores de los móviles y que puede ser adaptado según las necesidades.

Por último, mencionar la dificultad de organizar un proyecto de estas dimensiones y la gratificación de ver los resultados.

En resumen, este ha sido un proyecto en el que he disfrutado trabajando y aprendiendo cosas nuevas, tanto en aspectos de conocimientos como aspectos de desenvoltura a la hora de enfrentar un problema real y dar una solución al mismo aportando mis conocimientos tecnológicos.

4.2 Líneas de continuación del proyecto

4.2.1 Servicio Web

Con respecto al servicio las posibles mejoras serían:

- Autentificación en la cabecera HTTP, si no se accede con una autentificación se deniega la consulta. O utilizar token: cuando se hace login correctamente el servidor nos devuelve un token (valor idealmente único e imposible de falsear) A partir de este momento para cualquier operación restringida debemos enviar el token en la petición. También pudiendo usarlos con HTTPS para codificar los datos.
- Posibilidad de obtener graficas que representen la evolución en un sensor concreto respecto al tiempo de un paciente.
- Gestión de Sesiones Clínicas: para ofrecer una forma alternativa de almacenar, gestionar y crear sesiones clínicas, bibliográficas, teóricas, etc., tan importantes en la práctica médica de hoy día.
- Aumentar la información de la tabla de pacientes para almacenar toda la información general necesaria: alergias, fumador, historial clínico, etc.

- Posibilidad de recoger imágenes de los pacientes para casos en los que se necesiten.

REFERENCIAS

[1] «Deloitte - España, cuarto de los países desarrollados en penetración de smartphones,» [En línea]. Available: http://www2.deloitte.com/es/es/pages/technology-media-and-telecommunications/articles/Nota-prensa-consumo-medios-espana.html

[2] Vicente Pardo (2005), «Trastornos cognitivos en la esquizofrenia-Estudios cognitivos en pacientes esquizofrénicos: puesta al día,» [En línea]. Available: http://www.mednet.org.uy/~spu/revista/jun2005/04_edm_02.pdf

[3] Howden-Chapman, et al (1999), « Housing and health in older people: ageing in place, » Social *Policy Journal of New Zealand*, p 14-30.

[4] Mobyen Uddin Ahmed, et al. (2015), « Intelligent Healthcare Services to Support Health Monitoring of Elderly,» *Internet of Things. User-Centric IoT*, p 178-186.

[5] Ilsun You, et al. (2014), «Intelligent healthcare service based on context inference using smart device,» Soft *Computing,* p 2577-2586.

[6] «El android libre - ¿Cuáles son y para qué sirven los sensores de nuestros Android?,» [En línea]. Available: http://www.elandroidelibre.com/2014/07/cuales-son-y-para-que-sirven-los-sensores-de-nuestros-android.html

[7] «Página oficial de desarrolladores Android - Motion Sensors,» [En línea]. Available: https://developer.android.com/guide/topics/sensors/sensors_motion.html?hl=es

[8] «ICare,» [En línea]. Available: http://www.icarefit.com/

[9] «Página oficial de desarrolladores Android - android.Location,» [En línea]. Available: https://developer.android.com/reference/android/location/package-summary.html

[10] « Android,» [En línea]. Available: https://www.android.com/intl/es_es

[11] «El android libre - Android nunca ha estado tan presente como hoy en España,» [En línea]. Available: http://www.elandroidelibre.com/2016/06/android-cuota-ventas-moviles-espana.html

[12] «Hoy Android - El 7,5% de los dispositivos Android ya están actualizados a Marshmallow,» [En línea]. Available: http://www.hoyandroid.com/el-75-de-los-dispositivos-android-ya-estan-actualizados-a-marshmallow/

[13] « XAMPP,» [En línea]. Available: https://www.apachefriends.org/es/index.html

[14] «Apache,» [En línea]. Available: https://httpd.apache.org/

[15] «MYSQL,» [En línea]. Available: https://www.mysql.com/

[16] «Android Studio,» [En línea]. Available: https://developer.android.com/studio/index.html

[17] «Advanced REST Client,» [En línea]. Available: https://advancedrestclient.com/

[18] «PHP,» [En línea]. Available: http://www.php.net/

[19] «JSON,» [En línea]. Available: http://www.json.org/

[20] «DbDesigner,» [En línea]. Available: https://dbdesigner.net/

[21] «Visual Paradigm,» [En línea]. Available: https://www.visual-paradigm.com/

[22] «Hermosa programación - Realizar Peticiones Http Con La Librería Volley En Android,» [En línea]. Available: http://www.hermosaprogramacion.com/2015/02/android-volley-peticiones-http/

[23] «Página oficial de desarrolladores Android - android.Location,» [En línea]. Available: https://developer.android.com/reference/android/location/package-summary.html

[24] «Página oficial de desarrolladores Google - API de Google,» [En línea]. Available: https://developers.google.com/maps/?hl=es

[25] «Página oficial de desarrolladores Android - Location Strategies,» [En línea]. Available: https://developer.android.com/guide/topics/location/strategies.html?hl=es

[26] UkJae Ryu, et al. (2013), «Adaptive Step Detection Algorithm for Wireless Smart Step Counter, » *Information Science and Applications (ICISA), 2013 International Conference*, pp. 1- 4. [En línea]. Available: https://www.computer.org/csdl/proceedings/icisa/2013/0602/00/06579332.pdf

[27] Agata Brajdic, et al. (2013), «Walk Detection and Step Counting on Unconstrained Smartphones,» *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing,* 225-234 [En línea]. Available: http://www.cl.cam.ac.uk/~ab818/StepDetectionSmartphones.pdf

[28] «Página de proyectos de Google - Pedometer,» [En línea]. Available: https://code.google.com/archive/p/pedometer/

ANEXO A: CÓDIGOS PHP DEL SERVICIO

A.1. Conexión.php

El fichero conexión.php presenta los datos de acceso a la base de datos, usuario, contraseña, nombre de la base de datos y servidor donde se encuentra. E incluye la clase *Respuesta* que trata los mensajes de respuesta del servicio.

```php
<?php

define('hostname', 'localhost');
define('user', 'root');
define('password', 'FnteJPNwHLXCGj3R');
define('databaseName', 'tfg');
try{
$connect = mysqli_connect(hostname, user, password, databaseName);
}catch (mysqli_sql_exception $e){
    //si no se puede realizar la conexión
    http_response_code(500);
    exit;
}
class Respuesta{
function response($code=200, $status="", $message=""){
    http_response_code($code);
    if(!empty($status) && !empty($message)){
        $response = array("status"=> $status, "message"=>$message);
        echo json_encode($response, JSON_UNESCAPED_UNICODE);
    }
}
}
?>
```

A.2. comprobarUsuario.php

El fichero comprobarUsuario.php presenta el código utilizado para comprobar si existe un usuario y la contraseña es correcta.

```php
<?php
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


    mostrarFilas();
}else{
    $respuesta->response(400);
}

function mostrarFilas()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["contrasena"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];
    $contrasena = $_POST["contrasena"];

    $query = "Select * FROM usuario WHERE dni = '".$dni."' AND contrasena = '".$contrasena."';";
    $result = mysqli_query($connect, $query);
    $numero_filas = mysqli_num_rows($result);

    $temp_array = array();

    if($numero_filas > 0) {
        $respuesta->response(200,"success","usuario existe");
    }else{
        $respuesta->response(200,"error","usuario no existe");
    }

    mysqli_close($connect);
    }

}
?>
```

A.3. comprobarPaciente.php

Este método es utilizado para saber si un usuario es un paciente o no.

```php
<?php

require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


    mostrarFilas();
}else{
    $respuesta->response(400);
}


function mostrarFilas()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
    $dni = $_POST["dni"];

    $query = "Select usuario_dni FROM paciente WHERE usuario_dni = '".$dni."';";
    $result = mysqli_query($connect, $query);
    $numero_filas = mysqli_num_rows($result);

    $temp_array = array();

    if($numero_filas > 0) {
        $respuesta->response(200,"success","usuario existe");
    }else{
        $respuesta->response(200,"error","usuario no existe");
    }


    mysqli_close($connect);
    }

}
?>
```

A.4. insertarAcelerometro.php

Este método inserta los datos obtenidos del acelerómetro en su tabla correspondiente.

```php
<?php
/**
 * Insertar una nueva fila de acelerometro en la base de datos
 */
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


    createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["fecha"]) || !isset($_POST["eje_x"]) || !isset($_POST["eje_y"])
    || !isset($_POST["eje_z"]) || !isset($_POST["aceleracion"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $fecha = $_POST["fecha"];
    $eje_x = $_POST["eje_x"];
    $eje_y = $_POST["eje_y"];
    $eje_z = $_POST["eje_z"];
    $aceleracion = $_POST["aceleracion"];
    $dni = $_POST["dni"];

    $query = "Insert into acelerometro(Fecha,Eje_x,Eje_y,Eje_z,Aceleracion,usuario_dni)
    values ('$fecha','$eje_x','$eje_y','$eje_z','$aceleracion','$dni');";

    mysqli_query($connect, $query) or die (mysqli_error($connect));
    mysqli_close($connect);
    $respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.5. insertarUsuario.php

Este método inserta los datos para crear un nuevo usuario, tanto en la tabla de usuarios, como en la de médico o paciente según lo indicado.

```php
<?php
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["nombre"]) || !isset($_POST["contrasena"]) || !isset($_POST["fecha_nacimiento"])
        || !isset($_POST["usuario"])  || !isset($_POST["sensores"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];
    $nombre = $_POST["nombre"];
    $contrasena = $_POST["contrasena"];
    $fecha_nacimiento = $_POST["fecha_nacimiento"];
    $usuario = $_POST["usuario"];
    $medico_dni = $_POST["medico_dni"];
    $especialidad = $_POST["especialidad"];
    $email = $_POST["email"];
    $telefono = $_POST["telefono"];
    $sensores = $_POST["sensores"];
    $latitud_geofence = $_POST["latitud"];
    $longitud_geofence = $_POST["longitud"];

    $query = "INSERT into usuario(dni,contrasena) values ('$dni','$contrasena')";
    mysqli_query($connect, $query) or die (mysqli_error($connect));

    $query = "INSERT into sensores(lista_sensores_disponibles,usuario_dni) values ('$sensores','$dni')";
    $result = mysqli_query($connect, $query) or die (mysqli_error($connect));
```

```php
    if ($usuario == 'Paciente') {
        $query = "INSERT into paciente(usuario_dni,nombre,fecha_nacimiento,medico_dni,latitud_geofence,longitud_geofence)
        values ('$dni','$nombre','$fecha_nacimiento','$medico_dni','$latitud_geofence','$longitud_geofence')";
        mysqli_query($connect, $query) or die (mysqli_error($connect));
    }else{
        $query = "INSERT into medico(usuario_dni,nombre,fecha_nacimiento,especialidad,email,telefono)
        values ('$dni','$nombre','$fecha_nacimiento','$especialidad','$email','$telefono')";
        mysqli_query($connect, $query) or die (mysqli_error($connect));
    }
        mysqli_close($connect);
    $respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.6. mostrarDatosFecha.php

Este método muestra los datos de cualquiera de las tablas de los sensores del paciente correspondiente después de la fecha indicada.

```php
<?php
error_reporting(E_ALL);
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {

        mostrarFilas();
}else{
    $respuesta->response(400);
}

function mostrarFilas()
{
    global $connect;
    $respuesta = new Respuesta();
    if( !isset($_POST["dni"]) || !isset($_POST["dato"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        mysqli_query($connect, "SET NAMES 'utf8'");
        $Fecha = $_POST["fecha"];
    $dni = $_POST["dni"];
    $dato = $_POST["dato"];

    if($dato == 'Sensores'){
        $query = "Select * FROM ". $dato ." WHERE usuario_dni = '". $dni ."';";
    }else{
        $query = "Select * FROM ". $dato ." WHERE Fecha >= '". $Fecha ."' AND usuario_dni = '". $dni ."';";
    }
    $result = mysqli_query($connect, $query);
    $numero_filas = mysqli_num_rows($result);

    $temp_array = array();
```

```php
    if($numero_filas > 0) {
        while ($row = mysqli_fetch_assoc($result)) {
            $temp_array[] = array_map('utf8_encode', $row);
        }
    }

    header('Content-Type: application/json');
    echo json_encode(array($temp_array), JSON_UNESCAPED_UNICODE);
    mysqli_close($connect);
    }

}
?>
```

A.7. insertarLocalización.php

Este método inserta los datos obtenidos de la localización en su tabla correspondiente.

```php
<?php
header("Content-Type: text/html;charset=utf-8");
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


        createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["fecha"]) || !isset($_POST["latitud"]) || !isset($_POST["longitud"])
        || !isset($_POST["altitud"]) || !isset($_POST["precisionn"]) || !isset($_POST["pais"])
    || !isset($_POST["estado"]) || !isset($_POST["ciudad"]) || !isset($_POST["codigo_postal"])
    || !isset($_POST["direccion"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        mysqli_query($connect, "SET NAMES 'utf8'");
        $Fecha = $_POST["fecha"];
        $Latitud = $_POST["latitud"];
        $Longitud = $_POST["longitud"];
        $Altitud = $_POST["altitud"];
        $Precisionn = $_POST["precisionn"];
        $Pais = $_POST["pais"];
        $Estado = $_POST["estado"];
        $Ciudad = $_POST["ciudad"];
        $Codigo_postal = $_POST["codigo_postal"];
        $Direccion = $_POST["direccion"];
        $dni = $_POST["dni"];

        $query = "Insert into localizacion(Fecha,Latitud,Longitud,Altitud,Precisionn,Pais,Estado,Ciudad,Codigo_postal,Direccion,usuario_dni)
        values ('$Fecha','$Latitud','$Longitud','$Altitud','$Precisionn','$Pais','$Estado','$Ciudad','$Codigo_postal','$Direccion','$dni');";

        mysqli_query($connect, $query) or die (mysqli_error($connect));
        mysqli_close($connect);
        $respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.8. insertarAudio.php

Este método inserta los datos obtenidos del modo de audio en su tabla correspondiente.

```php
<?php
/**
 * Insertar una nueva fila de audio en la base de datos
 */

require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


        createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["fecha"]) || !isset($_POST["modo_audio"])  || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $Fecha = $_POST["fecha"];
        $Modo_audio = $_POST["modo_audio"];
        $dni = $_POST["dni"];

        $query = "Insert into audio(Fecha,Modo_audio,usuario_dni) values ('$Fecha','$Modo_audio','$dni');";

        mysqli_query($connect, $query) or die (mysqli_error($connect));
        mysqli_close($connect);
        $respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.9. insertarHumedad.php

Este método inserta los datos obtenidos del sensor de humedad en su tabla correspondiente.

```php
<?php
/**
 * Insertar una nueva fila de humedad en la base de datos
 */

require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


        createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["fecha"]) || !isset($_POST["humedad"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $Fecha = $_POST["fecha"];
        $Humedad= $_POST["humedad"];
        $dni = $_POST["dni"];

        $query = "Insert into humedad(Fecha,Humedad,usuario_dni) values ('$Fecha','$Humedad','$dni');";

        mysqli_query($connect, $query) or die (mysqli_error($connect));
        mysqli_close($connect);
        $respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.10. insertarLlamada.php

Este método inserta los datos obtenidos del estado de llamada del dispositivo en su tabla correspondiente.

```php
<?php
/**
 * Insertar una nueva fila de llamada en la base de datos
 */

require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


    createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["fecha"]) || !isset($_POST["modo_audio"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $Fecha = $_POST["fecha"];
        $Modo_llamada = $_POST["modo_audio"];
        $dni = $_POST["dni"];

        $query = "Insert into llamada(Fecha,modo_llamada,usuario_dni) values ('$Fecha','$Modo_llamada','$dni');";

        mysqli_query($connect, $query) or die (mysqli_error($connect));
        mysqli_close($connect);
        $respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.11. insertarLocTrabajo.php

Este método es utilizado para añadir la localización del trabajo de un médico.

```php
<?php

require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


        createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["latitud"]) || !isset($_POST["longitud"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];
    $latitud_trabajo = $_POST["latitud"];
    $longitud_trabajo = $_POST["longitud"];


    $query = "UPDATE medico SET latitud_trabajo = '".$latitud_trabajo."' WHERE usuario_dni = '".$dni."'";
    mysqli_query($connect, $query) or die (mysqli_error($connect));

    $query = "UPDATE medico SET longitud_trabajo = '".$longitud_trabajo."' WHERE usuario_dni = '".$dni."'";
    mysqli_query($connect, $query) or die (mysqli_error($connect));


    mysqli_close($connect);
$respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.12. insertarLuz.php

Este método inserta los datos obtenidos del sensor de luz en su tabla correspondiente.

```php
<?php
/**
 * Insertar una nueva fila de luz en la base de datos
 */


require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


    createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["fecha"]) || !isset($_POST["luz"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $Fecha = $_POST["fecha"];
        $Light = $_POST["luz"];
        $dni = $_POST["dni"];

        $query = "Insert into sensor_de_luz(Fecha,Light,usuario_dni) values ('$Fecha','$Light','$dni');";

        mysqli_query($connect, $query) or die (mysqli_error($connect));
        mysqli_close($connect);
        $respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.13. insertarPantalla.php

Este método inserta los datos obtenidos del estado de la pantalla en su tabla correspondiente.

```php
<?php
/**
 * Insertar una nueva fila de pantalla en la base de datos
 */


require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


    createFila();
}else{
    $respuesta->response(400);
}
function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["fecha"]) || !isset($_POST["modo_pantalla"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $Fecha = $_POST["fecha"];
        $modo_pantalla = $_POST["modo_pantalla"];
        $dni = $_POST["dni"];

        $query = "Insert into pantalla(Fecha,modo_pantalla,usuario_dni) values ('$Fecha','$modo_pantalla','$dni');";

        mysqli_query($connect, $query) or die (mysqli_error($connect));
        mysqli_close($connect);
        $respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.14. insertarPasos.php

Este método inserta los datos obtenidos del número de pasos en su tabla correspondiente.

```php
<?php
/**
 * Insertar una nueva fila de pasos en la base de datos
 */


require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


        createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["fecha"]) || !isset($_POST["pasos"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $Fecha = $_POST["fecha"];
        $Pasos = $_POST["pasos"];
        $dni = $_POST["dni"];

        $query = "Insert into pasos(Fecha,Pasos,usuario_dni) values ('$Fecha','$Pasos','$dni');";

        mysqli_query($connect, $query) or die (mysqli_error($connect));
        mysqli_close($connect);
        $respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.15. insertarPresión.php

Este método inserta los datos obtenidos del sensor de presión en su tabla correspondiente.

```php
<?php
/**
 * Insertar una nueva fila de presion en la base de datos
 */


require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


        createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["fecha"]) || !isset($_POST["presion"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $Fecha = $_POST["fecha"];
        $Presion = $_POST["presion"];
        $dni = $_POST["dni"];

        $query = "Insert into presion(Fecha,Presion,usuario_dni) values ('$Fecha','$Presion','$dni');";

        mysqli_query($connect, $query) or die (mysqli_error($connect));
        mysqli_close($connect);
        $respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.16. insertarProximidad.php

Este método inserta los datos obtenidos del sensor de proximidad en su tabla correspondiente.

```php
<?php
/**
 * Insertar una nueva fila de proximidad en la base de datos
 */


require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


        createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["fecha"]) || !isset($_POST["aproximacion"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $Fecha = $_POST["fecha"];
        $Aproximacion = $_POST["aproximacion"];
        $dni = $_POST["dni"];

        $query = "Insert into proximidad(Fecha,Aproximacion,usuario_dni) values ('$Fecha','$Aproximacion','$dni');";

        mysqli_query($connect, $query) or die (mysqli_error($connect));
        mysqli_close($connect);
        $respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.17. insertarTemperatura.php

Este método inserta los datos obtenidos del sensor de temperatura en su tabla correspondiente.

```php
<?php
/**
 * Insertar una nueva fila de temperatura en la base de datos
 */


require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


        createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["fecha"]) || !isset($_POST["temperatura"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $Fecha = $_POST["fecha"];
        $Temperatura= $_POST["temperatura"];
        $dni = $_POST["dni"];

        $query = "Insert into temperatura(Fecha,Temperatura,usuario_dni) values ('$Fecha','$Temperatura','$dni');";

        mysqli_query($connect, $query) or die (mysqli_error($connect));
        mysqli_close($connect);
        $respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.18. insertarTransicion.php

Este método inserta los datos obtenidos de la transición realizada en su tabla correspondiente (entrar o salir del perímetro prefijado).

```php
<?php
/**
 * Insertar una nueva fila de transición en la base de datos
 */


require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


        createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["fecha"]) || !isset($_POST["transicion"]) || !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $Fecha = $_POST["fecha"];
        $Transicion = $_POST["transicion"];
        $dni = $_POST["dni"];

        $query = "Insert into transicion(Fecha,Transicion,usuario_dni) values ('$Fecha','$Transicion','$dni');";

        mysqli_query($connect, $query) or die (mysqli_error($connect));
        mysqli_close($connect);
        $respuesta->response(200,"success","Nueva  fila insertada");
    }
}
?>
```

A.19. modificarMedico.php

Este método es utilizado para modificar el email de un médico mediante la sentencia UPDATE.

```php
<?php
/**
 * modificar un médico en la base de datos
 */


require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


        createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset(!isset($_POST["email"]) || !isset($_POST["dni"]))){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];
        $email = $_POST["email"];

        $query = "UPDATE medico SET email = '".$email."' WHERE usuario_dni = '".$dni."'";

        mysqli_query($connect, $query) or die (mysqli_error($connect));

        mysqli_close($connect);
        $respuesta->response(200,"success","Fila modificada");
    }
}
?>
```

A.20. modificarPaciente.php

Este método es utilizado para modificar los umbrales de temperatura, humedad y pasos de un paciente mediante la sentencia UPDATE.

```php
<?php
/**
 * Modificar un paciente en la base de datos
 */

require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


        createFila();
}else{
    $respuesta->response(400);
}

function createFila()
{
    global $connect;
    $respuesta = new Respuesta();
    if( !isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];
    $temperatura_max = $_POST["temperatura_max"];
    $temperatura_min = $_POST["temperatura_min"];
    $humedad_max = $_POST["humedad_max"];
    $humedad_min = $_POST["humedad_min"];
    $pasos = $_POST["pasos"];
```

```php
    $query = "UPDATE paciente SET temperatura_max = '".$temperatura_max."' WHERE usuario_dni = '".$dni."'";
    mysqli_query($connect, $query) or die (mysqli_error($connect));

    if($temperatura_min != ''){
    $query = "UPDATE paciente SET temperatura_min = '".$temperatura_min."' WHERE usuario_dni = '".$dni."'";
    mysqli_query($connect, $query) or die (mysqli_error($connect));
    }
    if($humedad_max != ''){
    $query = "UPDATE paciente SET humedad_max = '".$humedad_max."' WHERE usuario_dni = '".$dni."'";
    mysqli_query($connect, $query) or die (mysqli_error($connect));
    }
    if($humedad_min != ''){
    $query = "UPDATE paciente SET humedad_min = '".$humedad_min."' WHERE usuario_dni = '".$dni."'";
    mysqli_query($connect, $query) or die (mysqli_error($connect));
    }
    if($pasos != ''){
    $query = "UPDATE paciente SET pasos = '".$pasos."' WHERE usuario_dni = '".$dni."'";
    mysqli_query($connect, $query) or die (mysqli_error($connect));
    }


    mysqli_close($connect);
    $respuesta->response(200,"success","Fila modificada");
    }
}
?>
```

A.21. mostrarAudio.php

Este método obtiene el último dato incluido referente a un médico en la tabla del modo de audio, usado para mandar notificaciones.

```php
<?php
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {

        mostrarFilas();
}else{
    $respuesta->response(400);
}

function mostrarFilas()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{

        $dni = $_POST["dni"];

        $query = "SELECT Modo_audio FROM audio WHERE usuario_dni = (SELECT medico_dni FROM paciente WHERE usuario_dni = '". $dni ."')";
        $result = mysqli_query($connect, $query);
        $numero_filas = mysqli_num_rows($result);

        $temp_array = array();

        if($numero_filas > 0) {
            while ($row = mysqli_fetch_assoc($result)) {
                $temp_array[] = $row;
            }
        }

        header('Content-Type: application/json');
        echo json_encode(array($temp_array[$numero_filas-1]));
        mysqli_close($connect);
    }
}
?>
```

A.22. mostrarEmailTelefono.php

Este método obtiene el email y el teléfono de un médico, usados para mandar notificaciones.

```php
<?php
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


    mostrarFilas();
}else{
    $respuesta->response(400);
}

function mostrarFilas()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];

        $query = "SELECT email,telefono FROM medico WHERE usuario_dni = (SELECT medico_dni FROM paciente WHERE usuario_dni = '". $dni ."')";
        $result = mysqli_query($connect, $query);
        $numero_filas = mysqli_num_rows($result);

        $temp_array = array();

        if($numero_filas > 0) {
            while ($row = mysqli_fetch_assoc($result)) {
                $temp_array[] = $row;
            }
        }

        header('Content-Type: application/json');
        echo json_encode(array($temp_array[$numero_filas-1]));
        mysqli_close($connect);
        }
}
?>
```

A.23. mostrarUltimaLocalizacion.php

Este método obtiene la última latitud y longitud de la tabla de localización referente a un médico, usada para mandar notificaciones.

```php
<?php
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


    mostrarFilas();
}else{
    $respuesta->response(400);
}

function mostrarFilas()
{
    global $connect;
$respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];

        $query = "SELECT Latitud,Longitud,Fecha FROM localizacion WHERE usuario_dni = (SELECT medico_dni FROM paciente WHERE usuario_dni = '". $dni ."')";
        $result = mysqli_query($connect, $query);
        $numero_filas = mysqli_num_rows($result);

        $temp_array = array();

        if($numero_filas > 0) {
            while ($row = mysqli_fetch_assoc($result)) {
                $temp_array[] = $row;
            }
        }

        header('Content-Type: application/json');
        echo json_encode(array($temp_array[$numero_filas-1]));
        mysqli_close($connect);
    }
}
?>
```

A.24. mostrarLlamada.php

Este método obtiene el último dato referente a un médico en la tabla del estado de llamada, usado para mandar notificaciones.

```php
<?php
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST')

    mostrarFilas();
}else{
    $respuesta->response(400);
}

function mostrarFilas()
{
    global $connect;
$respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];

        $query = "SELECT modo_llamada FROM llamada WHERE usuario_dni = (SELECT medico_dni FROM paciente WHERE usuario_dni = '". $dni ."')";
        $result = mysqli_query($connect, $query);
        $numero_filas = mysqli_num_rows($result);

        $temp_array = array();

        if($numero_filas > 0) {
            while ($row = mysqli_fetch_assoc($result)) {
                $temp_array[] = $row;
            }
        }

        header('Content-Type: application/json');
        echo json_encode(array($temp_array[$numero_filas-1]));
        mysqli_close($connect);

    }
}
?>
```

A.25. mostrarLocGeofence.php

Este método obtiene la latitud y longitud del perímetro en el que se controlarán las transiciones de un paciente.

```php
<?php
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {



        mostrarFilas();
}else{
    $respuesta->response(400);
}
function mostrarFilas()
{
    global $connect;
$respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];

    $query = "SELECT latitud_geofence,longitud_geofence FROM paciente WHERE usuario_dni =  '". $dni ."'";
    $result = mysqli_query($connect, $query);
    $numero_filas = mysqli_num_rows($result);

    $temp_array = array();

    if($numero_filas > 0) {
        while ($row = mysqli_fetch_assoc($result)) {
            $temp_array[] = $row;
        }
    }

    header('Content-Type: application/json');
    echo json_encode(array($temp_array[$numero_filas-1]));
    mysqli_close($connect);
    }

}
?>
```

A.26. mostrarNombresPacientes.php

Este método obtiene todos los pacientes asignados a un médico, para imprimir una lista de ellos.

```php
<?php
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {

        mostrarFilas();
}else{
    $respuesta->response(400);
}

function mostrarFilas()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];

    $query = "Select nombre,usuario_dni FROM paciente WHERE medico_dni='".$dni."'";
    $result = mysqli_query($connect, $query);
    $numero_filas = mysqli_num_rows($result);

    $temp_array = array();

    if($numero_filas > 0) {
        while ($row = mysqli_fetch_assoc($result)) {
            $temp_array[] = $row;
        }
    }

    header('Content-Type: application/json');
    echo json_encode(array($temp_array));
    mysqli_close($connect);
    }

}
?>
```

A.27. mostrarPacientes.php

Este método es utilizado para saber si un usuario es un paciente o un médico.

```php
<?php

require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {

        mostrarFilas();
}else{
    $respuesta->response(400);
}

function mostrarFilas()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
    $dni = $_POST["dni"];

    $query = "Select usuario_dni FROM paciente WHERE usuario_dni = '".$dni."';";
    $result = mysqli_query($connect, $query);
    $numero_filas = mysqli_num_rows($result);

    $temp_array = array();

    if($numero_filas > 0) {
        $respuesta->response(200,"success","usuario existe");
    }else{
        $respuesta->response(200,"error","usuario no existe");
    }

    mysqli_close($connect);
    }

}
?>
```

A.28. mostrarPantalla.php

Este método obtiene el último dato referente a un médico en la tabla del estado de la pantalla, usado para mandar notificaciones.

```php
<?php
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {

        mostrarFilas();
}else{
    $respuesta->response(400);
}

function mostrarFilas()
{
    global $connect;
    $respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{

        $dni = $_POST["dni"];

    $query = "SELECT modo_pantalla FROM pantalla WHERE usuario_dni = (SELECT medico_dni FROM paciente WHERE usuario_dni = '". $dni ."')";
    $result = mysqli_query($connect, $query);
    $numero_filas = mysqli_num_rows($result);

    $temp_array = array();

    if($numero_filas > 0) {
        while ($row = mysqli_fetch_assoc($result)) {
            $temp_array[] = $row;
        }
    }

    header('Content-Type: application/json');
    echo json_encode(array($temp_array[$numero_filas-1]));
    mysqli_close($connect);
    }
}
?>
```

A.29. mostrarProximidad.php

Este método obtiene el último dato referente a un médico en la tabla de proximidad, usado para mandar notificaciones.

```php
<?php

require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {

        mostrarFilas();
}else{
    $respuesta->response(400);
}

function mostrarFilas()
{
    global $connect;
$respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];

    $query = "SELECT Aproximacion FROM proximidad WHERE usuario_dni = (SELECT medico_dni FROM paciente WHERE usuario_dni = '". $dni ."')";
    $result = mysqli_query($connect, $query);
    $numero_filas = mysqli_num_rows($result);

    $temp_array = array();

    if($numero_filas > 0) {
        while ($row = mysqli_fetch_assoc($result)) {
            $temp_array[] = $row;
        }
    }

    header('Content-Type: application/json');
    echo json_encode(array($temp_array[$numero_filas-1]));
    mysqli_close($connect);
    }
}
?>
```

A.30. mostarHumedad.php

Este método obtiene los valores de humedad máximos y mínimos de un paciente definidos como umbrales para las notificaciones.

```php
<?php
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


    mostrarFilas();
}else{
    $respuesta->response(400);
}
function mostrarFilas()
{
    global $connect;
$respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];

    $query = "SELECT humedad_max,humedad_min FROM paciente WHERE usuario_dni =  '". $dni ."'";
    $result = mysqli_query($connect, $query);
    $numero_filas = mysqli_num_rows($result);

    $temp_array = array();

    if($numero_filas > 0) {
        while ($row = mysqli_fetch_assoc($result)) {
            $temp_array[] = $row;
        }
    }

    header('Content-Type: application/json');
    echo json_encode(array($temp_array[$numero_filas-1]));
    mysqli_close($connect);
    }

}
?>
```

A.31. mostrarTemperatura.php

Este método obtiene los valores de temperatura máximos y mínimos de un paciente definidos como umbrales para las notificaciones.

```php
<?php
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


        mostrarFilas();
}else{
    $respuesta->response(400);
}
function mostrarFilas()
{
    global $connect;
$respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];

    $query = "SELECT temperatura_max,temperatura_min FROM paciente WHERE usuario_dni =  '". $dni ."'";
    $result = mysqli_query($connect, $query);
    $numero_filas = mysqli_num_rows($result);

    $temp_array = array();

    if($numero_filas > 0) {
        while ($row = mysqli_fetch_assoc($result)) {
            $temp_array[] = $row;
        }
    }

    header('Content-Type: application/json');
    echo json_encode(array($temp_array[$numero_filas-1]));
    mysqli_close($connect);
    }

}
?>
```

A.32. mostrarPasos.php

Este método obtiene el número de pasos de un paciente definido como umbral para las notificaciones.

```php
<?php
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


        mostrarFilas();
}else{
    $respuesta->response(400);
}
function mostrarFilas()
{
    global $connect;
$respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];

    $query = "SELECT pasos FROM paciente WHERE usuario_dni =  '". $dni ."'";
    $result = mysqli_query($connect, $query);
    $numero_filas = mysqli_num_rows($result);

    $temp_array = array();

    if($numero_filas > 0) {
        while ($row = mysqli_fetch_assoc($result)) {
            $temp_array[] = $row;
        }
    }

    header('Content-Type: application/json');
    echo json_encode(array($temp_array[$numero_filas-1]));
    mysqli_close($connect);
    }

}
?>
```

A.33. mostrarLocTrabajo.php

Este método obtiene la latitud y longitud del trabajo del médico.

```php
<?php
require 'conection.php';
$respuesta = new Respuesta();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {


    mostrarFilas();
}else{
    $respuesta->response(400);
}
function mostrarFilas()
{
    global $connect;
$respuesta = new Respuesta();
    if(!isset($_POST["dni"])){
        $respuesta->response(422,"error","Algún parametro vacio");
    }else{
        $dni = $_POST["dni"];

    $query = "SELECT latitud_trabajo,longitud_trabajo FROM medico WHERE usuario_dni =
    (SELECT medico_dni FROM paciente WHERE usuario_dni = '". $dni ."')";
    $result = mysqli_query($connect, $query);
    $numero_filas = mysqli_num_rows($result);

    $temp_array = array();

    if($numero_filas > 0) {
        while ($row = mysqli_fetch_assoc($result)) {
            $temp_array[] = $row;
        }
    }

    header('Content-Type: application/json');
    echo json_encode(array($temp_array[$numero_filas-1]));
    mysqli_close($connect);
    }
}
?>
```